

Farhan Sajid Rahman Masters project

by Farhan Rahman

Submission date: 07-Sep-2022 11:58PM (UTC+0100)

Submission ID: 185791390

File name: 10853_Farhan_Rahman_Farhan_Sajid_Rahman_Masters_project_430701_950776080.docx
(993.35K)

Word count: 17304

Character count: 89145



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Static code analysis to help teach and produce 'clean' and 'better' code from recommended improvements

by

Farhan Sajid Rahman

MSc Software Engineering

Supervisor: Tim Bashford

7th September 2022

DECLARATION

I, Farhan Sajid Rahman declare that I am the sole author of this Project; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Signed : 

Date : 7/09/2022

FORM OF CONSENT

I Farhan Sajid Rahman hereby consent that my Project, submitted in candidature for the M.Sc. Applied Computing degree, if successful, may be made available for inter-library loan or photocopying (subject to the law of copyright), and that the title and abstract may be made available to outside organisations.

Signed : 

Date : 7/09/2022

Table of Contents

Static code analysis to help teach and produce 'clean' and 'better' code from recommended improvements	0
Table of Contents.....	2
Table of Figures.....	3
Table of Tables	3
Abstract.....	5
Chapter 1 – Introduction	5
1.1 Background and History	6
1.2 Aims and Objectives.....	8
1.2.1 Expected Outcome	8
Chapter 2 - Literature Review	9
2.1 Static Analysis Techniques	9
2.1.1 Data flow analysis –	10
2.1.3 Abstract Syntax Tree	Error! Bookmark not defined.
2.1.4 Regular Expression	Error! Bookmark not defined.
2.1.6 Conclusion of Static Analysis Techniques	Error! Bookmark not defined.
2.2 Static Analysis Tools	10
2.3 Static Analysis Specification and Practice.....	12
2.4 Static Analysis Error Detection	13
2.5 Static Analysis Advantages and Disadvantages	14
2.5.1 Conclusion of Advantages and Disadvantages	16
2.6 Dynamic Code Analysis	16
2.7 Manual Code Analysis.....	17
2.8 Literature Review Conclusion.....	Error! Bookmark not defined.
Chapter 3 – Methodology.....	18
3.1 Research methodology	18
3.2 Development Methodology.....	19
Chapter 4 – Design & Implementation	21
4.1 Design	21
4.1.1 Programming Language and Software	21
4.1.2 Application Features	22
4.1.3 Application Design	23
4.2 Implementation.....	23
4.2.1 Building the User Interface	23
4.2.2 JavaFX.....	24
4.2.3 Eclipse IDE	24

4.2.3 Intentional Low-Level Programming Errors	24
Chapter 5 – Results and Discussion	25
5.1 SpotBugs	25
5.1.1 SpotBugs Conclusion	26
5.2 Checkstyle	27
5.2.1 Checkstyle Conclusion	28
5.3 Something	29
Chapter 6 – Conclusion	30
6.2 Recommendations	30
Chapter 7 – Reflection	32
8. Appendices	34
8.1 Appendix A – Design	34
8.1.1 Eclipse	34
8.1.2 JavaFX SceneBuilder	34
8.1.3 Flowchart of the Application	35
8.1.4 User Interface Storyboards	36
8.2 Appendix B – Implementation Stage	38
8.2.1 JavaFX Library	38
8.2.2 User interface component functionality	38
8.2.3 Object Oriented Programming Concepts	39
9. References	41

Table of Figures

Figure 1: Control Flow Graph Example [14]	10
Figure 2: Feature Driven Development Methodology	19
Figure 3: SpotBugs Errors Detected	25
Figure 4: SpotBugs Bug Description	25
Figure 5: SpotBugs Bug 2 Description	26
Figure 6: SpotBugs Developed Program Error	26
Figure 7: Checkstyle Coding Violation Display	27
Figure 8 Checkstyle violations displayed in classes	28
Figure 9 Checkstyle lowercase class name violation message	28

Table of Tables

Table 2: Categorised Static Analysis Tools	12
Table 1: Static Analysis Error Detection [33]	13

Keywords: Static Code Analysis, Static analysis, Dynamic Code Analysis, Dynamic analysis, NIST, SATE, FDD, CWE, SANS, OOP

Abstract

Software can be of varying complexities and a never-ending cycle for additions and improvements, it is always growing. It progresses into a codebase that can become a challenge to maintain and deal with on a regular basis. Direct example being with NASA having up to eight million lines of code for one of many of their projects. In the basis of errors and maintainability to validate and audit software code, the use of static analysis tools was introduced for serving a job to analyse code for properties that range from providing feedback on security vulnerabilities, error checking of bugs found, and the style of code written. The research through the document partook in assessing static code analysis with the theory of the technology being a tool that can be used with the intentions of learning, and a tool that can drive developers to produce 'better' and 'cleaner' code from feedback it produces. In referral to 'better' and 'cleaner,' it can be subjective, but it refers to code in compliance with coding standards that follows specifications and practices of appropriate quality. Examples of standards being incorporated with static analysis tools are with MISRA, CWE and SANS where each serve a strong set of guidelines in the practice of creating code in different programming languages and objectives. The direction in accomplishing the idea of the project was through reviewing literature and conducting a test. The first static analyser Lint was published in 1978, and so the literature read and tools available builds up from a large period of years with the topic still showing to be relevant in address to the use cases, concerns, and benefits. One prominent and persistent issue since the initial tool of Lint being released is the case of static analysis tools detecting false positives, enough to make developers stray away from making use in practice as found in questionnaires done by other authors. Static analysis researchers have made conscious effort and investigation in solving the issue, and though it can be eased, it is an inevitable result. There are a vast number of studies and research made by others, and it made it necessary to see what results can be found from assessing the research in contrast with the aims and objectives that are made of the project. The intentions head to see a didactic side to static analysis, with plan of gathering information surrounding the topic and making idea if an automated analysis tool can provide benefit in learning and producing code. The chosen static analysis tools to create the tests included FindBugs, CheckStyle, CheckThis and CheckThat. The overall results made in evaluation of all tools had displayed positive results in terms of being instructive and teaching toward a developer, though with connection to the opinions and information gathered from the literature review, they were in accordance with the found information that false positives can be an issue. With all of the information digested, it causes for an unclear conclusion on whether Static Analysis can truly be for teaching.

Chapter 1 – Introduction

This project goes onto conducting a thorough study of static code analysis, also known as static analysis. It is a tool which acts as a debugger that runs before an execution of a program and outputs a review of the code that was checked [1]. The specific area throughout this paper will be focused on the target of finding answers to the question in which the title poses; if static code analysis provides a source of teaching and delivering good recommendations to produce 'clean' and 'better' code. The idea was inspired from a suggestion made by the supervisor of this project in the past, "Structured code analysis to grade code against best practice specifications and recommend improvements." When further researching around that title that was suggested, it had led into uncovering static analysis and forming the path of this project.

The idea and the problem deliver interest as it surrounds software engineering, it questions if a tool like this provides a benefit to the programmer in the aspect of learning and making better decisions during the development of their code. If static analysis brings an aspect of teaching to the developers using such a tool, it can be assumed that better habits and arrangement of code would be made in return while avoiding stated bugs. Since the process is automated, it can be thought that not all issues may be identified and may even detect code that have no issue but reports it. In relation to that assumption, further research will be done in the literature review chapter and can take a look at the consistency of static analysis detecting issues as it would further demonstrate whether it would be good to learn from. Having a tool that brings automation for the process of analysing code and collecting feedback, this project intends to seek into the effectiveness of the results and determine if it can provide teaching developers and aiding them in to composing better and cleaner code and seeing if it is suited with making use of appropriate standards, good coding specifications and practices. In addition to the research taking place, it can bring light to the necessity of having a static analysis tool as part of software development practice and regular use. After the literature has been assessed around the related topics involved with static analysis and the project question, a later chapter will be planned to discuss results from trying and reviewing a range of static analysis tools with a program that is designed and implemented. This is for intentions of gathering empirical data from how effective static analysis tools work in the direction of what was stated at the beginning, if developers can learn from them to produce 'clean' and 'better' code. With the program that is being built, the contents of what it provides as an application will not be of importance, but the amount of code that is written to conduct greater tests of the static analysis tools will be necessary as planned in the later methodology chapter.

1.1 Background and History

The given purpose of static code analysis is to automatically scan and identify the quality of written code amongst a set of rules before being ran. It reviews and results in recommending improvements that can be made. With the varying tools available such as a static analysis tool named FindBugs, from analysis it provides a review of code with ranking of the severity. It highlights the defects in distinct colours which are red, yellow, and green to represent high, medium, and low [2]. Conversely, it can be said that developers can manually analyse code, however, in regard to the time it takes, the automation of static code analysis gives the capability of pinpointing mistakes and errors in a faster pace than manually looking through and understanding lines of varying complexity of code. To also mention, with manual analysis it could result in possibilities of human error due to easily missing something while examining through the written lines of code. A context for the use of a tool like this can be where developers and teams are in pressure for a review to determine standards are being met from their code, as well as identification of weaknesses in their program [3]. Moreover, it can be mentioned that static analysis allows for quality assurance teams to validate and have an approach to detecting vulnerabilities in what the development team may have mistakenly produced [4]. It especially becomes of high demanded use when codebases become large and complex. It can be assumed that it will be difficult to understand as well as time consuming for a team to manually maintain and make changes to a large software for improvement [5].

It is found that static analysis is one of two types of a program analysis technique, with dynamic analysis being the second. To reiterate what was mentioned in the introduction, static analysis is the process of analysing code before an execution of a program, it is completed with advanced algorithms and methods [6]. Whereas dynamic analysis is a method of analysing code that completes during the runtime of a program, and this is to

primarily warn and give metrics for issues such as memory leaks, resources used and more of which is associated while the software is running, such as execution time of the built features in the program. The objective behind a program analysis tool is to provide information that was analysed about the codebase of a program being developed, where then a returning set of results would be given to display properties of which is related to error checking and security related issues of the composed code [6] [7].

A paper by Patrick Thompson gives an introduction into static analysis and discusses a historical perspective, where given information states that the first static analysis tool was made in the 1970s along with the rise in more modern software development techniques. A tool that went by Lint created by Stephen C. Johnson was the first static analysis tool built and released from 1978-1979 with the purpose of detecting programs made in the C language for errors. It is said that the C compilers during that period of time did not perform as many correctness checks, and so a solution the Lint tool brought was to analyse and bring more error checks such as stating warnings amongst variables that were not used and initialised, suspicious type conversions, and more of which are within C compilers today, but however still held to be of use and important during that time as it filled in a role that assisted in the making of C programs. This information in relation to the project highlights that static code analysis tools assists a developer in a way to realise their mistakes and can be presumed that it assists to be aware and avoid from making the same mistake again. Lint was proven to be of influence and created a movement as it brought in the name of 'linters' for referring to the different modern static analysis tools created [5]. A prominent team in the world who make use static code analysis is NASA, the national aeronautics and space administration team. They have obligations to have continuous service in the code they create as they have a large codebase for their technology and creations. With one being over eight million lines of code, it is stated that just making a change and any sort of bug fixes bring risk of more underlying issues. It is said that it would be impossible for NASA to evaluate that much of a large volume of code, so they must make use of a method for the improvement of their software quality [8].

1.2 Aims and Objectives

To make mention, this section provides the aims and objectives that the project plans to hold and be achieved through the literature and understanding gained from the research that will be completed.

The overarching aim of this project is to:

- To find that static code analysis helps with the teaching and appliance of development standards and practices for 'clean' and 'better' code

The objectives based off of the overarching aim of the project:

1. To establish a strong understanding for the topic area and find the importance of static analysis. This will be measurable through completion of research and deemed realistic as resources such as journals can be found online, as well as suitable available books and retrieving information from those who are familiar with the topic. The time required for completion would be up to two weeks of thorough investigation of static analysis.
2. To investigate existing tools in line with the project aim that can be looked at and tested to gain knowledge of how they work, and evaluate personal experience with available static analysis tools created on a program that is built and contains a suitable sized codebase
3. To critically evaluate the testing that was done with static analysis tools and discuss the analysis feedback and whether it seems appropriate in helping teach and push to write 'better' and 'cleaner' code
4. To evaluate all relevant information found and determine what static analysis has to offer in the context of the project. This is achievable as it will be done after all relevant areas have been looked into and discussed and allow for a shorter timeframe to finish due to having the information at hand.

1.2.1 Expected Outcome

The outcome that is to be expected from the research is to find out that static analysis effectively supports developers and teams in learning and producing cleaner as well as good quality code in the sense that the static analysis tool reviews it over the best specifications and coding practices.

Chapter 2 - Literature Review

Throughout the sections of the literature review, it will be looking at a mix of reliable sources including academic journals, books and overall creditable research completed by others who have done work in relation with static analysis and the path of this project. The sections will take part in discussing and distinguishing key information that can assist in solving and forming the path to complete the aims and objectives.

2.1 Static Analysis Techniques

As part of the literature review, the first aspect of static analysis explored is a look into the techniques that static analysis tools are built from. This is in purpose to provide an understanding on how static analysis tools can differ from one to the other and establish an awareness of the techniques that static analysis researchers and developers have implemented. The research can be used to point toward identifying the range of complexity between methods and also give an understanding to lead the project into categorising them fittingly when in the testing phase.

To start the research of techniques, it began with a conference paper from Michael Striewe and Michael Goedicke providing a review of static analysis approaches for programming exercises, where they focus on instructive static analysis feedback for students learning [9]. A section of the conference paper goes into comparing the different approaches in static analysis, the assessment made makes contrast of a source code and byte code approach, then trees and graphs. Source code is the instruction of code that builds a program, it is written by the developer in normal and readable form that the machine cannot read. For the machine to make an understanding of programming languages, it must be converted into machine code, however with a language like Java, it makes conversion from source code to intermediate level code which is bytecode. Every time the program is ran, the compiler will generate a bin folder containing the classes in binary form, which would be the byte code. It is mentioned and said that the execution speed in comparison, byte could would execute quicker and performs higher due to being in machine code form [9] [10]. The authors throughout this paper write in a learning standpoint of static analysis and brings up valuable information these methods bring to static analysis. To continue, they make comparisons that both techniques bring capabilities of offering syntactical checks, examples being the visibility of knowing how many and what types of fields there are, as well as structures of inheritance usage throughout the program. The comparison then makes mention of unnecessary code statements, where for optimization a compiler may remove such code. Though it sounds like a benefit for unneeded code to be removed, it is said that in the aspect of learning it can be seen as a disadvantage. In result of a compiler removing the code, a static analysis tool scanning through byte code cannot report the statements, whereas source code analysis will without the limitations in place because the code can be viewed from the source directly. In conclusion, it is said by the authors that in scenarios of being taught from the method of static analysis with bytecode, it does not fill all requirements, whereas static analysis with source code does.

From the research of Moritz Belle, Radjino Bholanath and more, it is said the traditional method used for static analysis tools is with data-flow analysis [11]. Further research makes mention of even more techniques of which are implemented in different tools, involving Regular Expression and Abstract Syntax Tree (AST) [12].

2.1.1 Data flow analysis –

Continuing the research from Moritz, data flow analysis is stated to be the top technique that is used for static analysis. The procedure of the technique starts with creating a control flow graph, the process begins with the structuring of a graph represented as the program. For each node that is in link with the graph are held with equations written, and the equations in question will be continuously solved to provide a calculated output from given input at each of the nodes for until there is a stable point reached. Further research by Peng Li makes mentioned that data flow analysis brings purpose for establishing and identifying the use of variables when compiled. In mention of the control flow graph aspect used, it provides purpose for investigating if there is a value assigned with the vulnerability of the variables [13].

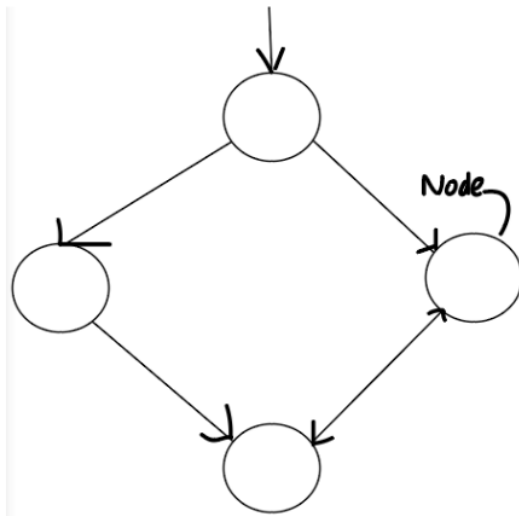


Figure 1: Control Flow Graph Example [14]

2.2 Static Analysis Tools

Part of the aims and objectives of the project is to make test of the different available static analysis tools for gathering empirical data and information of how they work. The program is planned to make use of Java, and so the tools selected will require support of this programming language. This is to see the benefits that can be taken advantage of in relation to the project goal. Before looking at tools, research made display that there are a number of techniques which different tools can make use of, the available static analysis tools can lead onto this section being mentioned by their technique and for later making choice of what tools can be used for the testing. Making test of several tools of which use the same technique to accomplish the analysis would not be as insightful, and so it will be essential to select variety as it would deliver a broader collection of results from the tests.

To begin the research of static analysis tools, the upbringing of the variety of tools can be mentioned from the information referenced in a paper titled "Why Do Software Developers Use Static Analysis Tools?" [15]. As mentioned briefly in the document already, static analysis tools began being used in the 1970s, and this paper makes mention that it was for primarily optimising the compiler but then expanded toward the use for finding bugs, security vulnerabilities and even went into code automation which is linked to refactoring and code completion. The bigger analysis tools began evolving during the early 2000s with the objective being for detecting bugs, tools mentioned were Fortify and Coverity. The industry

continued on getting progressively getting bigger for the purpose of cutting costs from improved analysis and detection of bugs.

In a paper by Andrei, it is made clear that there are numerous commercial as well as open-source tools available for static analysis, each having different claims about efficiency along with performance. For practical use, the best tool for static analysis should ideally have the strongest detection for errors and vulnerabilities, and least number of false positives [16]. The commercial industry has taken ahead in the building of advanced static analysis tools which are expensive, and there are companies that have taken initiative of creating their own in-house built static analysis software, example being Microsoft PRefix. Other companies also made mention of that has deployed and makes use of a static analyser is Facebook with Facebook Infer [17]. The Infer tool was made to be continuously running for verifying all code modifications made in the Facebook application for android and iOS. The languages supported are Java, C and Objective-C which is the language used for OS X and iOS [18].

For finding state of the practice static analysis tools and their effectiveness, Static Analysis Tool Exposition (SATE) is a study that holds non-competitive intentions to improve tools and raise awareness to the public. Analysis is done by the National Institute of Standards and Technology (NIST) where the developers of a static analysis tool would use their analyser on different programs and have the results forwarded and investigated by NIST to later publish findings [19]. Also provided by NIST is a large table listing many different available commercial and free tools, involving description of their abilities, what language they offer to work with and date of new additions [20]. With use of this information, it allows for choosing good static analysers that can be used for testing purposes within the project to allow for an evaluation of use.

One of the static analysis tools included was FindBugs, this is a tool included in the list by NIST and one that was commonly mentioned in various research papers [21] [22] [23] and a look into the website of the tool makes a statement that it is an open-source tool with the ability to detect bug patterns [24]. These patterns were said to be implemented using a bytecode analysis library called BCEL, and it was found that this library serves the purpose of making modifications toward java classes with the ".class" format which would be the binary classes. BCEL gives the possibility to create and manipulate the java binary classes and analyse them. The library is used in a number of projects of which were successful and includes analysis tools being one of them [25]. David and Pugh who are the authors of "Finding bugs is easy" [21], provide additional information stating that the bug detectors make use of a design pattern called Visitor, and the visitor design pattern is where the detector makes a visit of each class of the program. Implementation strategies involved with FindBugs are of what was mentioned in earlier discussion of static analysis techniques. The paper says FindBugs makes use of dataflow analysis and being mentioned is that the method is the most complicated for handling data flow and control flow. In another paper, FindBugs is said to be beneficial for making analysis of the issues found in designs and any violation to style and coding standards [22]. The last FindBugs version was last released in 2015, and a successor named SpotBugs forked the project and is continuing being built with the community. This in result has the capabilities of the FindBugs tool with improvements added, such as up to 400 patterns of bugs capable of being detected [20].

The next tool as part of the list in the NIST, was Checkstyle. From the official Checkstyle website, It is a tool that supports the Java language for static analysis and states that it is ideal for projects in wanting to administer a coding standard. As the name makes it clear, this static analysis tool makes goal of analysing code and ensuring the style is in

strength of a coding standard being followed. It also contains ability to find issues in reference to design, layout, and formatting. The example configuration files included as part of Checkstyle were Sun Code Conventions and Google Java Style which are the Google standards of writing code in Java that follows appropriate and strong guidelines of code formatting, naming and general practices that should be followed [26] [27].

From the different tools found and discussed, it was categorized in table 2 to highlight the language it supports, the technique it uses and the objective it serves when being used. This then leads onto choosing the tools that will be selected during the testing phase and running each of them in a program that will be built. The idea is to select each tool to provide a different path to review how they differ and highlight a sense of teaching observed from the results they output with relation to their objective.

Table 1 : Categorised Static Analysis Tools

Tool	Language	Analysis Technique	Objective
FindBugs	Java	Data-flow Analysis (Bytecode)	Detect bugs
SpotBugs	Java	Data-flow Analysis (Bytecode)	Detect Bugs (FindBugs Fork)
Facebook Infer	Java, C, Objective C	Separation logic and bi-abduction	Verify Code modifications
Checkstyle	Java	Unknown	Assess program code quality and design

For the procedure of testing, an appropriate range of tools can be selected from the research investigated. In purpose to avoid equivalent results and to effectively make test of the project aims and objectives, the decision was made to avoid selecting static analysis tools with similarity of delivering the same objective, such as two tools with focus on bug finding. The decision made is to choose different paths to get a wider perspective on what feedback can be resulted in the different objectives and to see how it is suitable in teaching the developer in diverse ways. The path chosen was to pick one for detecting bugs, the second to detect coding style issues and the third for security problems. The tools of which were chosen are appropriate to make use of and review as part of the testing to see how the results are in the aspect of learning and being taught. To begin the selection, FindBugs was chosen to test as it can provide feedback that is related to faults and bugs of the program. Alongside FindBugs, a second selection that can be made is Checkstyle.

2.3 Static Analysis Specification and Practice

In the aim of the project, it can be questioned how static analysis could provide reliable feedback that can teach a developer good practice and producing 'better' code. Investigation was done to see what measures are made to give effective and dependable comments in relation to the source code scanned.

As found, for a static analysis tool to give suitable feedback on a software implementation made, it carries a coding standard to follow to provide valuable information in the outcome of the results of the code to the developer. A tool can have a pre-defined and formatted requirements from an existing coding standard that it follows when grading the code, an example mentioned in this white paper is called MISRA, where static analysis can review code in the standards and practices which are part of MISRA and have the written code scanned and verify it in context of being MISRA compliant [28]. MISRA is a coding

standard and states that in the development of critical software systems, it is a widely used guideline for quality assurance of code and for assuring any written code is in line with requirements that are part of the specification. MISRA can be seen as a popular mention for a specification and has included a statement that it is a guideline that gets refined, and also had gone through multiple re-adjustments for years. Moreover, the guidelines should be applied and planned from the beginning of a software being built, as if the code is not already MISRA compliant, it is likely that it follows little of the practices involved, meaning it would result in rewriting and a possible whole re-work of the code structure just to be aligned with the specification [29]. In further research, there are more mentions of other created coding standards implemented with static analysis which are CWE and SANS, both of which stand for Common Weakness Enumeration (CWE) and Sysadmin, Audit, Network and Security (SANS) [30].

2.4 Static Analysis Error Detection

In software engineering bugs can be easily overlooked depending on varying factors, large development projects especially due to having a sizable amount of code written. It can be hard to pinpoint where the error may be without extensive debugging and manually searching. These statements can be backed by an occurrence in the past where in 1991, a mistyped character in a line of code prevented 12 million subscribers of AT&T, a phone service company from using their service because the network went out in effect of the minor mistake made [31]. In result of the error, it is believed that sixty million dollars were lost, and this amount does not include the business lost from companies relying on the network [32].

Research went further to look at what static analysis can achieve and be assistance of around the cases of various errors. A paper that was done by Anjana Gosain and Ganga Sharma had led into a compiled a list of which errors can be thrown while using a static analysis tool [33]. These types of errors were also spoken of in a research paper by Nachiappan and Thomas in works with Microsoft Research, stating them as "low-level programming errors" [34]. To best portray the information gathered, it was collated into Table 1 to display the list of errors it can detect and an explanation.

Table 2: Static Analysis Error Detection [33]

Error	Description / Example
Out of bound Arrays	When accessing an array out of the size initialized
Memory Leaks	Memory which the program does not return after temporary use
Nullpointer deference	Pointing to a value that is NULL
Buffer overflow and underflow	Overflow is when a buffer exceeds the capacity of storage [35]. Underflow is when the buffer accessing memory is read faster than it is being sent [36]
Invalid arithmetic	Errors thrown by arithmetic mistake; example being dividing a value by 0

Access of Uninitialized Variables	Variables which are not given a value
Infinite Loops	A loop that will not exit after a condition and constantly iterates
Non-Terminating Functions	When a program does not call back after a function ran

In judgement of the errors displayed and found, an additional element to the program can be to include deliberate low-level errors as listed in the table. This can be for the sake of advancing the outcome of results. The errors can just be in one folder of files but not have actual effect on the any functionality and the program built, it will purely be for the tool to include as apart of analysis to review the outcome.

2.5 Static Analysis Advantages and Disadvantages

Throughout many of the papers discussing static analysis, there are mentions of what attracts and prevents teams with the use of static analysis. One of the most common constraints identified in the research is that a static analysis tool can frequently produce a substantial number of false positives as mentioned in several different papers [6] [11] [37] [38], where it marks pieces of code serving as an error, though not actually being one. In these cases, it would bring the developers to have to manually fix the result given, and it is said to produce enough to be a deterrent from making use of static analysis tools in the development process. Static analysis researchers have made effort in creating solutions and improvements toward easing false positives and though it was encouraging results, it was stated that it still serves as an issue to be problematic [38]. As mentioned in the history in the, it was also a problem occurring when Stephen C. Johnson released the first static analysis tool called Lint in 1978-1979 [5].

In discussion by Par Emanuelsson and Ulf Nilsson [39], there were areas of which showed valuable advantages that static analysis tools can bring to validating software further. From their research it brought attention that static analysis has the possibility of making detection and or solutions to problems that a traditional compiler cannot detect. This is also said in reference to problems that may be of difficulty to manually make a test for. Before making highlight towards what the problems are, the writers make statement to say that static analysis does not validate a software being free from runtime errors. This can be found necessary to make mention of so that the act of a static analysis tool is not to be heavily depended on. The runtime errors mentioned that static analysis can detect is related with the errors spoken of in Table 1 of the research from Gosain and Sharma [33]. It brings advantage of seeing a general view of different errors being detected and made clear to the developer. An additional mention by Par and Ulf were with static analysis dealing with dead code and data, where the code is not made use of or reached in any way. Attention is then brought to stating that in the result of static analysis detecting defects, it can bring more to a developer producing efficient code. This was said in specific mention with Java, where efficiency is not the objective as it is a language that is considered safe.

In a study by Brittany Johnson, Yoonki Song and Emerson Murphy, it was titled "Why don't software developers use static analysis tools to find bugs?" [2] and part of their research had included acquiring twenty developers to interview and collect opinions from their use of static analysis tools. This included sixteen professional developers and four graduates with industry experience. The opinions and answers gathered were from an interview length of 60 minutes per person, and it resulted in the dilemma of seeing static

analysis as beneficial to use but also disliking the false positive outputs. In the mention of the tool output, fourteen out of the twenty made voice of negative impressions in regard to false positives. This was one of the main reasons that the developers who were questioned, neglected from using it as part of their software development. Issues that were also commented on was the problem of an abundance of warnings to look through and seeing it as “a bunch of junk” which expresses a lack of useful information given. Amongst all of the mentions of the negative aspects that were commented on, positives which were spoken of during the interview was that because static analysis is a tool that automates and serves such a process, it is a good thing and five participants had mentioned it should be worth using due to it delivering an automated method of performing a needed but mundane job. On another positive aspect, it touched on that development environments have inbuilt features of static analysis where three of the twenty developers made use of it due to its availability. In overview of the interview conducted by the researcher’s made justification to the issues behind static analysis and how much of a problem some of the results are and how it effects the overall use of the tool. Although seen as a useful tool in concept, the issues looked to overweigh any positives within the questionnaire.

One article on the topic of automated test data generation makes mentions of software testing being 50% of a software development budget, however, automated tools provide an advantage for reducing the costs. This can be expected because it no longer requires a person to work and do the given job the automated tool can now complete. The same study then goes onto mentioning different limitations static analysis holds, and one states that there are not enough trained staff that can accomplish using a tool to effectively perform static code analysis. It brings a point of the fact it may require workplaces to create the time to introduce staff training with a static analysis tool. Furthermore, the paper makes clear static analysis allows for finding vulnerabilities and errors in the written code but lacks the ability of doing so during the runtime of a program, dynamic analysis however does, but then the limitation is flipped where dynamic analysis cannot do what static analysis is capable of [40].

Another literature found is one that specifically evaluates the advantages and disadvantages by Liisa and Rain. In this paper, they had gone through a great quantity of publications researching advantages and disadvantages and giving the main information collated into tables to display them. The main advantages involved with static analysis tools is said to prove it assists in the understanding of software behaviour, detecting problems without the program running and brings capability of errors being detected early. Static analysis is also a tool that allows for detection of advanced common problems within programs such as buffer overflows, security issues and race conditions. The main disadvantages were said to be issues with false-positive and negative results, incapability of detecting all defects and also shared that different tools give different warnings making it difficult to understand [37].

In further literature, a conference paper in 2019 titled, “Explaining Static Analysis – A perspective” done by Marcus Nachtigall, Lisa Nguyen and Eric Bodden [41], thoroughly made clear of many of the usability challenges part of static analysis, as well as the design challenges it faces in building the tool. In the discussion it was firstly emphasised that the latest research conducted in the topic of usability in static analysis, highlighted the recurring problems of warnings and confusion of the findings from the tool. The results bring an unfavourable outcome and a bad user experience from static analysis tools made as of late. They went through collating the usability issues identified in publications spanning over a decade, and in general this brings additional clearance of the deterrence it has amongst developers. The issues are listed as challenges, and there were six challenges declared that

a static analysis tool has. The usability issues as stated from the beginning are warning messages, fix support, false positives, user feedback, workflow integration and specialized user interfaces. In summary of each challenge, it was found that the warning messages serve to be an issue when it comes to developers making the decision to fix it. In many of the static analysis tools they researched, the messages are said to be generic with a lack of detail displayed and it should be pushed for all tools to start becoming more informative and have it adapted throughout different static analysis tools. Fix support is in mention to the tool supporting the developer with offering methods of correcting an error, as the developer reading the error or warning in attempt to resolve may not have the knowledge behind it.

2.5.1 Conclusion of Advantages and Disadvantages

In overall conclusion of the positive and negative qualities found from different papers, the negative elements of static analysis tools were put on display more in comparison to the advantages it makes hold of. A static analysis tool is valued and recognised for the job it aims to serve, however, there is more improvement to be made and changes to be adapted to in order to have a range of static analysis tools of good standard. After the discussing the found and reviewed advantages and disadvantages, it brings question to if the found information will have a weighing with the testing that will be done and presented within chapter four. The same opinions and findings have a chance to be part of the output and it may depend on the tools chosen.

2.6 Dynamic Code Analysis

The opposite of static analysis is dynamic analysis where it runs during runtime in comparison to static analysis ran before the execution of the program, and although not in the aims of the project it can be mentioned of what it achieves in contrast to static analysis. The researchers of a paper titled "A Systematic Survey of Program Comprehension through Dynamic Analysis" [42], makes understanding that dynamic analysis provides a solution to ease the time it takes to gather program comprehension. It was said that the time it takes in understanding a software built, can be up to 60% of the effort taken in the production of the software. Dynamic analysis brings the aspect of highlighting the software during runtime. It makes display of the behaviour, and the picture it aims to portray is from the class-level details up to the higher-level architecture of the system. The advantage dynamic analysis has over static analysis is stated to being the information that is given during the execution of the program, bringing benefit to software following an object-oriented concept as it exposes the objects and can assist in resolving late bindings of functions. On the other hand, the disadvantage is that the results that are given, are only for the situations taken place while being dynamically analysed.

In the research of a paper overviewing static analysis, "An overview on the Static Code Analysis approach in Software Development" [43], the authors make discussion of dynamic analysis throughout it. It represents dynamic analysis as not a tool for directly identifying the problems of the written code composed by the developer, but the problems related to the behaviour of the code that was composed to make the software. The tool is intended for use in practice for printing the program faults and vulnerabilities to the developer. Although static analysis can be taken advantage of it does not mean that dynamic code analysis cannot be used. A combination of both can be encouraged for validating the program and code from defects detected from running the tools at separate times. It can be argued whether to assess the software with dynamic analysis first, or review the code with static analysis, as these two analysis tools provide as distinct systems between each other. The comparison between static and dynamic analysis is that static will overview the written code and make sure it follows compliance of standards and analyse any weaknesses that can be susceptible to issues. Dynamic on the other hand, makes analysis

of the running program and looks for irregularities throughout the time it is executed. In a closing statement of dynamic code analysis, the research makes grasp that it would not teach the developer how to produce better code, though make a display how the behaviour of their software will function and provide as a method of debugging.

2.7 Manual Code Analysis

Manual code analysis would be as the name suggests, the analysis of code for errors and changes to be made, all done manually. In the statements written by Brian Chess and Gary McGraw, the manual review of code would require the developers to have the knowledge in place already and have the idea of where to be looking in the first place within the source of code [44]. To relate this to the project, manual analysis of code would not be as beneficial to the teaching of creating better code as it would rely on experience and skills acquired for making a thorough review and it would in result be exhausting to make complete understanding of everything. An analysis done manually, would either be through a self-review of the source or a peer review. A self-review is the code reviewed by the writer to find and fix issues and assess that the written code is following good practice or could be done better. A peer review would be done by another person assigned the task, it could be part of the development team or by a third party [30]. The process can be a long and tedious job depending on the size of the software being worked on. There are codebases with up to eight million lines of code, as mentioned in the background section with NASA [8], where every minimal change and error fixing may just cause further issue. In a case like this, it would seem making it unfeasible for a team to be looking at a screen of lines of code they may not even understand. Directing back at the literature by Ivo, manual code analysis is stated as a time-consuming job and also supports the fact with the reviewer needing to know the code that they are reading means, as well as what type of errors they are trying to detect. It is easy for a person to miss something which then may cause problems within a workplace, but with a tool it can be precise looking at each line with of code performing the job at a faster rate and allow developers productivity to be increased with saved time to use in other areas [43]. It would be impossible for a self-review to be done on millions of lines of code on a regular basis of a changing software, static analysis would serve as a necessity for enhancing the speed, effort, and time [30].

To make additional comments on manual code analysis, Wagner, Jurjens, Koller and Trischberger had made an investigation that can be declared by the title of the paper, "Comparing Bug Finding Tools with Reviews and Tests" [23]. The investigation of the study went into the path of conducting tests of static analysis tools which they refer as bug finding tools. The tests done were with other methods of analysis for the goal of getting an understanding of both results, they acknowledged in the abstract the thought that automation with static analysis may reduce time spent in code review. The study was planned with use of five separate projects and making comparison to the findings of the tools with functional tests such as black box and white box testing alongside reviewing code. The outcome from the investigation had made statement that making use of static analysis tools proved to be consistent, though manual reviews was said to find more bugs in comparison to static analysis. They also made indication that the static analysis tools provided results that were missed during manual review. This makes evidence toward manual review having a risk of having faults missed, but also static analysis too. Although, only open-sourced tools were used and may need improving, or tests made with other stronger tools to see if it makes detection of the manual bugs found. In the discussion, it was said that both played in a beneficial aspect, as the tool allows for identifying and removing errors before the program is ran, then inspect.

Chapter 3 – Methodology

The methodology chapter heads into discussion of the different procedures, approaches and decision-making choices that were made to accomplish the path and goals of the project. The project serves both an experimental and developmental aspect, where there is a program to be developed and a hypothesis to be investigated if static analysis helps teach and produce 'better' and 'clean' code. This chapter will follow through engaging discussion of the procedures taken in order to complete the requirements.

3.1 Research methodology

In this project, it involves three approaches being a developmental, experimental and a document review. It served as an experimental as well as development approach due to the aim and objective of gathering empirical data of different static analysis tools and testing it with a developed program which will be built. Behind the idea of the project, it seeks an answer for finding out if static analysis can be a way to provide a method of teaching to produce improved code. The planned direction is to research and gather literature available around the topic, finding the necessary information relative to the aims and objectives and then performing a test with a built program. With an assortment of literature collected and read expanding from a range of years, it occurs to be necessary to test and make clear of the material that was interpreted. The information put together from research come from several types of sources which include academic research papers, conferences, books, journals, official documentation sites, and other sources which are deemed reliable as well as authors dedicated in related fields.

The project intends for gathering qualitative data where the data received will be the information taken from the research of the literature review, and then making use of the information to take it further with the testing and getting results. In mention of the decided approach to the project, it will be in an inductive manner where the goal is to develop a theory and answer it with use of the qualitative data of the information taken from overall research. Following from the statements made about research, the philosophy being followed is on the positivist direction, the information that is found and collated will be verifiable statements, and the research done is cooperatively independent and objective.

In the period of the testing phase to gather and later showcase in the chapter discussing results, results will come from making a test of a range of static analysis tools selected from research. The testing will involve using a static analysis tool to inspect the codebase of a program, where in this case the program will be designed and developed. This will be expanded in a dedicated section and chapter. With the code written of the program, the tool would make attempt to search for problems and faults to then provide a review with possible suggestions and fixes to be made. Furthermore, with concerns to keep the tests consistent, the same composed code for the program will be re-used throughout each of the static analysis tools. For the results looking to be achieved, qualitative observations with intent of staying in path of the project will be made during the testing. Some of the results to be gained from the testing entails in evaluating how significant the techniques differ from every tool being used, deciding whether it would be promising to make continuous use of the tool, and then more of which is in expectation of answering the project goal of whether this allows for learning from the tool and producing better code. In conclusion, the results that will be output is an overall assessment of the tools decided to make use of and will include connection to the literature as well as observations which refer to the project aims and objectives.

3.2 Development Methodology

As mentioned, there is going to be a task of testing static analysis tools. For these tools to be evaluated effectively and deliver useful results to examine, a program of a reasonable sized codebase must be developed and created so that each tool which is made use of, can scan a fair amount of code implemented to review. The goal is to have a runnable and functional program without any conscious errors made. In the direction of the program that is developed, the application features that it will contain would not be of importance for the reason being as the code is only being assessed in the direction of what the project is aiming for, finding the effectiveness and an answer to the title question. The code and the depth of code is what will be significant for the running of tests done on the different static analysis tools. Although conscious errors will not be made as part of the program, a section for static analysis errors was looked at during the literature review chapter. This chapter included a collection of different bugs that are seeming to be a standard for static analysis tools to detect them. An idea developed from the research is to have a folder as part of the program with a class of code involving a variety of low-level programming mistakes. It will contain errors in relation to the research and not to be detected by the compiler. The erroneous code will not be involved with the workings of the functionality of the program but brings value for it to be a part of the scanning and assessment through different static analysis tools to gather in more results from the feedback generated. It will bring purpose for getting a broader understanding of the error messages that can be reviewed. In justification of the idea, there are types of errors that may not be detected in just the code implemented of the program built. It brings further observation of static analysis tool feedback and allows more attention to be measured to whether if they are instructive and can be of benefit to a developer to learn from.

To go forward from what was said and to build a program that is of a decent size, features should be built in a fairly quick manner to allow for content to continuously be implemented. As the program is primarily aimed to have working code and up to three thousand lines of it **[need to re-find the paper]**, the building of features should have priority because having focus on different stages of development rather than the design and implementation would use up time that can be for writing code and building the application. As speed of development is what is required, the choice was decided to follow an approach which is part of the agile development framework methodology. The plan is to follow a feature driven development (FDD) cycle, and this approach will put the attention to be largely on the phases of designing and building the program [45]. This practice was chosen in preference to follow and always provide a minimum viable product that is functional with features in place. It then can allow for further features to be implemented continuously while the application is still being functional as needed until the times up.

The cycle of FDD contains five stages as shown in figure 1 [45].

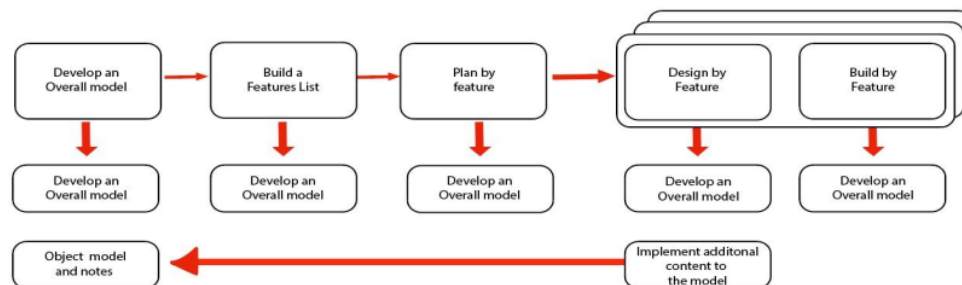


Figure 2: Feature Driven Development Methodology

The benefit of choosing an agile methodology of development proposes an aim toward a more productive and lighter approach of developing. The purpose is to have an iterative and incremental way of building software, and with FDD, the feature part of it is an important aspect with objective to provide development projects in time. In part of the disadvantages for FDD is that there is no guidance given toward the management of requirements for gathering and analysing it [46]. To work around the disadvantage revealed, methods of support can be involved in taking part of easing the issue. To distinguish the requirements to implement before development of the project, flow chart diagrams and the use of storyboards can assist to provide visual representation and identify the necessary ideas of features to be worked on. The more working code being added, the more results that can be reviewed by a static analysis tool.

Chapter 4 – Design & Implementation

This next chapter follows on from the methodology chapter and takes the discussion of the design and implementation stage further and holding more emphasis of the strategies taken to accomplish creating the program. This includes emphasising how it will be in line with achieving the intended goal. Throughout the design and implementation section, it will be walking through designing and implementing a developed program with a codebase that is suitable to be ran through different static code analysis tools. In the aims and objectives of the project, the end result is focused on testing written code with the static analysis tools, and so the centre point of discussion in the project is not around what features the program will have but to bring forward a good program to test and get results for the next chapter of investigating the aims and objectives made. This chapter will bring light of the design decisions and the implementation of the program and concepts that will be applied in anticipation of making significant tests of the static analysis tools. From the research during the literature review, these tools have the objective of detecting defects and outputting information about the code it reads. With this in mind, it sounds fitting to presume a program that is built and involves a good range of programming concepts, brings a wider scope of code to effectively make a test of. The code would be analysed with the various selected static analysis tools with the techniques that are integrated into them. The reason behind the presumption comes from reading how complex the tools can be with the algorithms in place for auditing software quality and assessing issues that were made in the program the tools make use of the complex techniques to accomplish what they target to do, it seems essential to evaluate them to an appropriate level in order to make use of the capabilities the tools have. This then allows for collecting even more valuable results and to make discussion of it in the next coming chapter of disclosing the outcome.

4.1 Design

The design section will be the planning of the application to be implemented. It will highlight the aspects of the program being made which would include the language, the software being used, a list of features and a visual display of what the program will look like and how it is intended to function with flowchart diagrams and a storyboard.

4.1.1 Programming Language and Software

In concern to the language that the program will be built upon, it has been decided with the choice of the Java programming language. This decision was made from the static analysis tools that were spoken of. It each had good compatibility with android languages, and the decision is also based off of having experience and knowledge with the Java programming language. Being experienced will allow for quicker production of the program and making tests of the static analysis tools that were chosen. In mention of software that will be used in the development of the program, one will be the Eclipse Integrated Development Environment (IDE) for java developers. It is a software that will work effectively due to the close support of Java, and it provides the needed tools with a simple workspace [47]. Also, within Eclipse, are the use of plugins which are pluggable components in purpose to provide different tools and features to incorporate in the development environment. There is a marketplace where different useful plugins can be searched and installed [48]. Part of the plugins within the marketplace also include different Static analyser tools [49] [50], this can be investigated for part of the testing phase and easily allow for gathering results of the program after the code has been implemented.

Alongside Java, the use of the JavaFX library and SceneBuilder tool will be highly used, it is integrated with eclipse [51] and this library provides a set of packages to provide the ability to design and create a functional graphical user interface (GUI) with buttons,

checkboxes, and many other components to make use of. One of the key features of JavaFX is the possibility to build a GUI with a supporting tool called SceneBuilder, where it allows the developer to visually design an interface by dragging and dropping the different available interface components. Once designed, it will then produce the JavaFX user interface application in an FXML file that can be linked within the Java code and JavaFX properties to make it work and be runnable. In summary, SceneBuilder will be for building the user interface, and the functionality of the buttons and features must be done through writing code to make everything operate as needed [52] [53]. This provides as an advantage because it saves time from tediously building the application interface with written code, it is easier to visually create and edit through the tool. In result, the functionality can be of main focus once designed. As seen in [8.1 appendix A](#), it shows the eclipse development environment of the program, the SceneBuilder application tool that will be used with the integration into eclipse, and then the format of an FXML file it produces after creating the interface.

4.1.2 Application Features

To discuss the concept and the design of the program that will be written and implemented, it will be based off of the logic of an order system. A list below will be given to display the primary features and what the order system plans to achieve:

- File system in place to create/delete/edit text files for the purpose of viewing information in text form. E.g., Customer details, Orders made
- Customer account creation / Customer logging in – On loading the application a customer will have option of creating an account or logging into their existing account
- Data persistence, to save necessary object files such as customer properties to be saved every run
- Ability of selecting products – The user can view and add or remove any products available into a checkout basket
- Purchasing Order – The user can go into viewing the order details of the products selected, and purchasing it to create the order
- Viewing orders made – The user can view the orders that they have created. Involve sorting algorithm to sort the list
- Cancelling orders – The users can select and delete any orders that they have created

From the features listed, these are the essential ones that should be mentioned and will be fulfilled throughout the implementation. The goal is to make a test of the code, and the decision behind the path is for creating a codebase that can include a variety of implemented code that can be reviewed. From the literature, the types of typical errors that are detected by static analysis tools were looked at. They were displayed in Table 1 and were stated as "low-level" programming errors, involving buffer overflows, buffer underflows, uninitialized variables and more [33] [34]. The intention of the project seeks to see if the tools can be useful for teaching, and so in the sake of gathering and analysing results when it comes to testing, the decision was made to include written code involving a number of the found errors but not have it part of the program functionality. The intent of this will be to bring more results and make assurance of seeing if it can detect and output appropriate and helpful feedback in order to solve such errors. The approach that this can be done is with having one class out of the way from the others, and having the code held with a variety of severe errors written but not classed as an error by the IDE and compiler.

4.1.3 Application Design

As seen in [8.1.3 appendix A](#), there is a display of a flowchart diagram created and is shown in two parts to visualise how the application will function and be like upon running. The flowchart was created in mind of the requirements and features that are planned to be within the application, it considers how the application will be ran and be functional. To further visualise the application, story boards were created to display how the user interface will be designed and function with the mentioned features that will be implemented in the program. When it comes to implementation, the design plan then can be followed and act as a guide to look at during the duration of building it. Storyboards of the user interface can be seen in [8.1.4 appendix A](#).

As mentioned in the beginning of the chapter, a requirement was put together for the inclusion of a variety of programming concepts when it comes to writing the code. In overview of the project path, the features which were spoken of are just to discuss how the application will be like. The main concern is what code will be involved to perform solid tests of the static analysis tools, and it is planned to involve several object-oriented programming concepts to be part of the program code being written. In java there a few of which can be listed and found from the oracle documentation website, the program will involve:

- Inheritance – Make use of a parent class to allow other classes to extend from it and re-use the same code [54]
- Interfaces – Make use of an interface to provide a blueprint for classes that can implement it [55]
- Abstraction [56]
- Polymorphism [57]
- Encapsulation [58]

From each of the listed concepts, the code will be constructed in the direction of applying each of them when it comes to building the code.

4.2 Implementation

This section will be of the implementation stage creating the program that will be assessed with static analysis tools to review how the code is assessed. Sections included will be the stages in how the application is made, this consists of creating the user interface, to adding functionality, making use of the different OOP concepts and the finished product.

4.2.1 Building the User Interface

The implementation began with building the user interface of the application. Once this is completed, it only leaves the features and functionality of the application to be focused on. A storyboard was created using Adobe Illustrator to portray a design that can be followed, it worked as a guide while creating the interfaces and it allowed for keeping productive and knowing the direction. The storyboard involved seven different interfaces of which allowed for navigation between one another with the use of buttons involved, and so each of the user interfaces will be built using the SceneBuilder tool. As mentioned in the design section, the SceneBuilder tool has the ability to be integrated with eclipse and it is what provided the ability to create each of the user interfaces. The process involved dragging and dropping components which was in accordance with the storyboard made, and the components that required adding functionality had an identity ID property attached for when it comes to writing the code functionality it is expected to serve. An example of a process can be a mouse click listener for the create account button, log in button and more. This can be seen in [8.2.1 appendix B](#) with given explanation to show the understanding of how components will have added functionality.

4.2.2 JavaFX

The Java development kit (JDK) used to hold the modules of JavaFX but in the recent years, it was made by decision from Oracle to have it separated within the new updated Java versions from 2018 onwards. It was recently extended support on Java 8 but only for the approaching years until March 2025 [59]. The current version of Java is coming to 19 in September 2022 [60], and so Java 8 would be eleven versions behind and will be incompatible in the near future due to no more support being given. The decision behind the removal was given a slight mention in a blog stating that it will allow for JavaFX to be an easier adoption while being available as a separate download from the JDK. With the removal from the JDK, it would now require importing the library to your project to make use of it. The technology of JavaFX library is now provided on the OpenJFX website where Oracle co-leads it [61] [62]. Before knowing to download and import the JavaFX to the project modules, this resulted in the code generated by Eclipse giving a number of errors due to JavaFX not belonging in the Java development kit anymore. It served as a slight problem as without it being part of the development kit, it has in effect made it a requirement for the JavaFX libraries to be built into the project module configurations for use of the JavaFX classes and JavaFX code, as well as to allow running the application. In further mention, to run the application as the modules were altered to support JavaFX, the run configuration VM arguments required an additional comment to provide the path of the module folder and JavaFX FXML module.

4.2.3 Eclipse IDE

With the Eclipse IDE, it was used to a high advantage to keep productive throughout the implementation of building the desired program. It served to do 0

The auto code completion shortcut made for a productive

As part of the Eclipse marketplace, there are a number of tools available in the marketplace, a portion of tools found being for the static analysis of code, of which were also part of the research of different tools. The decision

4.2.3 Intentional Low-Level Programming Errors

As mentioned in the methodology, in effort of gathering more results and to make the testing procedure stronger for the aims of the project, intentional errors were made within a class in a folder away from the core functionality of the program. This task was an additional idea to be part of the implementation, it was based off of a section of research made in the literature review in chapter 2 [33].

Within a file, the errors shown in table 1 within chapter two, had been implemented in order to see if the tools will pick it up and then display valuable information in ideas of fixing and or teaching.

Chapter 5 – Results and Discussion

This section goes onto discussing results that were achieved from scanning a designed and implemented program with different static analysis tools. The discussion will be from the observations made from use of the tool and be based around the information that is known about static analysis from the literature review. To ensure the discussion of results is on topic, it will stay in line with the proposed idea with the aims and objectives that were formed. A conclusion will be given to assess whether the aims and objectives were achieved from the tests made.

5.1 SpotBugs

As discussed in the literature review, SpotBugs is the successor of FindBugs and holds the same functionality as the tool but with implemented improvements. FindBugs is a tool that has stopped development in 2015, while SpotBugs is still being actively updated as a static analysis tool to find bugs [63]. For the use of the tool, it was capable through the eclipse marketplace plugins [64] bringing the benefit of using the integrated environment while looking at the detected potential issues found.

Once the program was scanned, as seen in figure 3, it made highlight of only four errors of which were all ranked "Scary" on the severity scale, indicating that the bug is of high risk and may require being fixed as soon as possible.

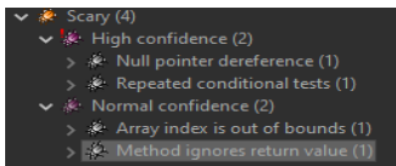
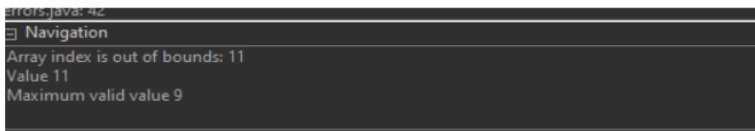


Figure 3: SpotBugs Errors Detected

Finding the file and investigating into these errors then made display of the description of the bug that was found. Figure 4 displays the issue and description. As part of the implementation stage, a class of deliberate errors were created just to assure output of bugs were made. One of the simple bugs was an out of bound array as found in the literature review of low-level programming errors.



Bug: Array index is out of bounds: 11

Array operation is performed, but array index is out of bounds, which will result in `ArrayIndexOutOfBoundsException` at runtime.

Rank: Scary (7), **confidence:** Normal

Pattern: RANGE_ARRAY_INDEX

Type: RANGE, **Category:** CORRECTNESS (Correctness)

Figure 4: SpotBugs Bug Description

In another instance of a bug that was detected by the static analyser, it was of a null pointer dereference. The explanation gives a clear display of what the bug is and shows the error it can lead to during execution of the code. Though these were intentional errors made, in the case of this being in the functionality of code, it can result in the program crashing with the nullpointer being returned as stated in the description. The ranking can be seen as suitable due to the potential case of crashing if it was used.

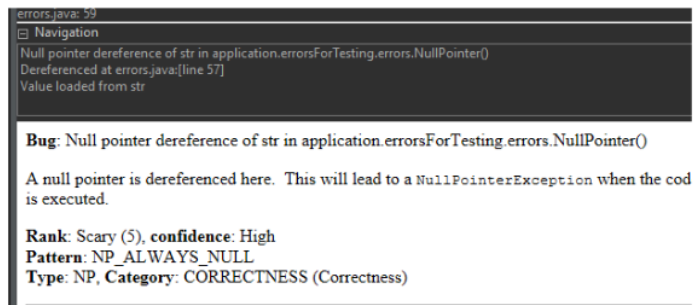


Figure 5: SpotBugs Bug 2 Description

There were only two bugs of which were detected in the source code of the actual implementation of the program, both of which were simple errors to fix and were overlooked during the building of the program. One of the errors that was made is shown in figure 6. SpotBugs made it clear of what the issue was and had guided me in the right direction to make the alterations to fix, where two of the same condition was being performed.

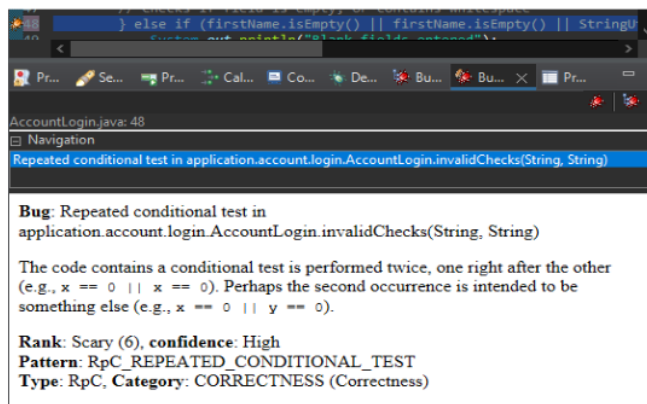


Figure 6: SpotBugs Developed Program Error

5.1.1 SpotBugs Conclusion

SpotBugs serves the job of finding bugs within a codebase that is scanned. From the detection of the program that was implemented, the only errors that were found in the main implementation files were only two of which were overlooked, and it had provided the fixes and attention of the errors being there. The other two errors were in a created class of planned errors in the case there were no results to be given. In overall review, the information that was displayed makes of decent detail toward the bug and provided fair example and suggestion within the feedback. In the aspect of teaching, spot bugs provided awareness of different bugs and pushes for more attention to be put forward when writing code.

5.2 Checkstyle

The second static analysis tool chosen was Checkstyle, and again, was available as a plugin provided in the Eclipse marketplace. This provided advantage of using it with the eclipse user interface and it makes it of ease to navigate and explore the warnings and errors that are output by the tool. To start making a scan of the project to see the coding violations of Checkstyle, it was simply done through a right click of the project source folder, and then selecting the option. Checkstyle has coding standards in configuration of which are the Sun Coding Conventions and Google Java Style coding standards [26]. The program designed and implemented did not make any use of the stated standards and so was output with an abundance of Checkstyle violations. Although, it allowed for seeing how the results would be collated together and whether it is done suitably. It also further brings in the investigation to view whether the solutions provided are explained well and easy to solve.

As the standard was not followed, there was a lot of feedback provided related to the practice in place for the formatting and styling of code. As seen in figure 3, there are a total of 3,071 violations of the coding practice, with a range of occurrences displayed in different files which can be seen into by selecting one of the warnings in the list. In judgement of the description of the feedback, as seen in the Checkstyle violation type section, it is explained in a reasonable manner that it gets the violation across, though a section that contains a simple solution message would improve and make it easier to deal with fixing violations. If the error can be detected, the question raises why a solution cannot be served. In positive note with the user interface, a feature that allows the violations to be easily navigated is that each of the items can be clicked in the box and get directly sent to the location of the issue in the certain file.

Overview of Checkstyle violations - 3071 markers in 29 categories (filter matched 3071 of 3071 items)

Checkstyle violation type	Occurrences
Wrong lexicographical order for 'X' import. Should be before 'X'.	38
'X' should be separated from previous line.	4
First sentence of Javadoc is missing an ending period.	59
Line is longer than X characters (found X).	75
Each variable declaration must be in its own statement.	8
'X' construct must use '{}'s.	8
'X' has incorrect indentation level X, expected level should be X.	536
'X' child has incorrect indentation level X, expected level should be one ...	1
Unused X tag for 'X'.	2
Abbreviation in name 'X' must contain no more than 'X' consecutive ca...	3
Array brackets at illegal position.	6
Javadoc comment is placed in the wrong location.	1
Summary javadoc is missing.	19
Comment has incorrect indentation level X, expected is X, indentation s...	2
Name 'X' must match pattern 'X'.	11
'X' is not preceded with whitespace.	2
'typecast' is not followed by whitespace.	15
At-clause should have a non-empty description.	13
Distance between variable 'X' declaration and its first usage is X, but allo...	2
Line continuation have incorrect indentation level, expected level shoul...	1
'X' child has incorrect indentation level X, expected level should be X.	531
Empty line should be followed by <p> tag on the next line.	3
'X' at column X should be on the same line as the next part of a multi-b...	2
'X' is not followed by whitespace.	5
Block comment has incorrect indentation level X, expected is X, indenta...	4
Javadoc tag 'X' should be preceded with an empty line.	41
Line contains a tab character.	1637
Extra separation in import group before 'X'	12
Missing a Javadoc comment.	30

Figure 7: Checkstyle Coding Violation Display

Further look at Checkstyle were the violations as shown in the files. Each line that has a violation is colour coded to highlight where the issues lay. This provides a benefit to be aware of where the issue is. Figure 4 displays the highlighting of lines showing violation and the message it provides. An improvement to be made is the possibility to add a message giving the solution to the violation. The tab character is in reference to the line being indented, and for the violation to be fixed, it wants the indentation to be removed. A solution being indicated to the developer may be of assistance in the case that they were not aware of what the description means, with further example being in figure 5.

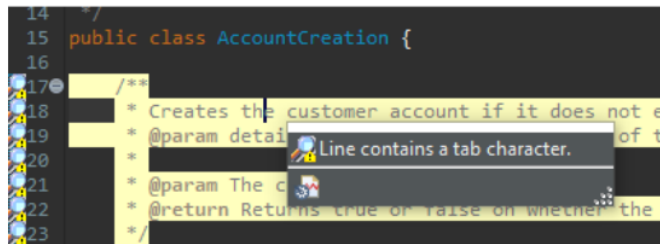


Figure 8 Checkstyle violations displayed in classes

The figure 5 displays a class error where it purposely violates the standard of class file naming. With it being all lowercased the Checkstyle tool detects it as an error and displays the message in an unclear way. In the aspect of teaching, a more distinct description to explain the issue should be in place with a given solution. This could be of more beneficial to make the developer aware, as well as knowing how to easily deal with the problem.

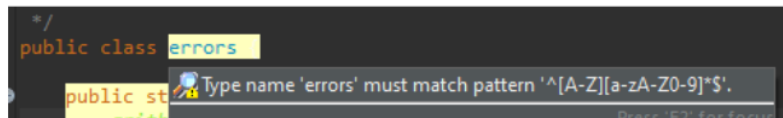


Figure 9 Checkstyle lowercase class name violation message

5.2.1 Checkstyle Conclusion

In conclusion of the Checkstyle static analysis tool, it displayed reasonable information of the violations in regard to the coding standard in place, however, raised concern in the explanation of the messages it gave. Although many are understandable, some are not clear enough and it can be improved by having a solution provided. The direction of the static analysis program serves to be useful with the ability of detecting the coding style issues, as it can make for good practice to teach a developer to follow. The enhancement of teaching and a developer learning to make better practices can be done from better given explanations, solutions, and examples. This way it can be of more beneficial and leading to how it should be done.

5.3 Something

Chapter 6 – Conclusion

- This chapter is where you tie up all the loose ends in the previous chapters. It is most important that it relates to what you have described previously, and that it does so in a relevant and concise fashion.
- In your summing-up, you need to show how what you did contributed to meeting the objectives you set in the introduction. In doing so, it is appropriate to repeat (in summary form) key points from your review, design, implementation, and evaluation as necessary.
- Any further work that can, or should, be undertaken to expand upon your work is to be highlighted, fully explained, and justified. The benefits gained from your work should be identified (these are **not** to include personal benefits). Also, any recommendations should be included here.

This chapter makes conclusion of the literature and findings of the research. Also, in relation to what was found and what was targeted in the aims and objectives of the project, recommendations will be discussed to justify what could be modified or added in order to strengthen the results.

To begin the conclusion of the project, it was found that static analysis certainly has the capability to teach better coding but is faulted by underlying issues which cannot be easily resolved. There are a range of techniques and levels of complexity in providing static analysis, however, false positives and unnecessary warnings have not been possible to be avoided. The research had shown that it is possible to ease the amount that is output, but it is still an issue and in result has built a barrier from some developers not finding it ideal to be used in practice. A reason behind the barrier is because it causes them more trouble than a feeling of assistance as stated by one of the people from an interview conducted. In the kinds of errors that static analysis can detect and notify, it is said to be good for low-level programming errors. The list of low-level programming errors stated were binary overflow, binary underflow, invalid arithmetic, infinite loops, non-terminating functions and more. The tests were completed of a program that was designed and then implemented with Java and JavaFX using the Eclipse IDE and the SceneBuilder tool. The developed program then led onto achieving empirical results from using different released and open-sourced static analysis tools looked at through studies and official websites. The purpose of the tests were to view the effectiveness of the results it can have in the eyes of a programmer seeking to learn from the feedback given. The results output from the tools that were chosen were insightful but to an extent due to the lack of tools that were assessed, the insightful information that was gathered is that it shows to have potential of teaching developers, but the tool has to be improved to give more thorough feedback which encourages the developer to learn. The explanations given by Checkstyle had not given more than one line of what the issue is, though SpotBugs was a more a hopeful tool which provided good explanations, a severity ranking and an example for the defect to be fixed.

6.2 Recommendations

For recommendations that could strengthen the path and direction of the project, an initial aim that was not included due to time constraints, was gathering more qualitative data through implementing questionnaires and or interviews as part of the methodology. Introducing these methods could have exhibited more information to make discussion of and allow the project question to further be interrogated. Involving this method could help discover more thoughts and results to analyse if static analysis can be more of a benefit than nuisance to use. The questionnaires or interviews can be done on developers who have experience using or having static analysis code as part of their practice in software

development. The questions in mind would be similar to a research paper that was found that had commenced interviews on 20 developers [2], it would be beneficial to ask if the developer in question if they have experience, find static analysis to be useful, if they believe it can teach a developer to make better choices, and more of which would be in line with the project title. For the testing phase of the project, time limitations were an issue again and the recommendation to be made is to make a test of more tools found from research and have more code implemented that can be reviewed by each of the tools. With the more code and features to make test upon, that will provide more resource to a static analysis tool to review and provide feedback to in the discussion. In result, it will allow for more data to analyse in mind of answering the question of the project path. As different static analysis tools are based on a programming language, a following recommendation can be to test a built software on various series of languages. This can allow for the results to be questioned and analyse if there are stronger tools to be used and provide more benefit in attention to the project depending on the different programming language used.

Chapter 7 – Reflection

To end the document, a reflection can be given to make understanding of improvements that could have been followed to strengthen the path and how it could be done again. With the goal in mind, it was not expected to see static analysis have such underlying problems which make it a deterrent for developers to use. The primary issue with static analysis tools found from research is that it can give a common occurrence of false positives where the detected code has no issue but was given in the feedback as a detected error. Furthermore, initial plans of this project were to contain more qualitative data involved with interviews and questionnaires to add additional strength of the research and results achieved to answer the project question further. In more concerns, a GANTT chart plan could be altered to provide more time and effort in different areas of static analysis. This project had aimed to involve more research around tools and techniques, however, the techniques looked at during research would be ranging of differing complexities and not enough time to make clear understanding of. To also include, time constraints proved to be an issue for finding quality tools of several types to make test of a built program. A lot of time was put into minor details of the program while the features implemented were not of importance. A program could have been built with a larger codebase if only the code was to worry about, rather than a user interface involved, although, it allowed me to further learn a technology of JavaFX and Scenebuilder with FXML files. In addition, from the two tools that were evaluated, a larger codebase could have been more beneficial to bring in more results. As part of the aim, at least three strong tools with each containing a different technique were to be tested, however, after doing two there and no time left for another, it should be more than three involved as the tool being tested may not generate as much feedback as seen with SpotBugs. It would seem to be beneficial to get a basis of more results to make the answer clearer if static analysis tools truly bring teaching and better produced code.

The last section of the report will focus on reflecting upon the project, the reason for this is to understand what could have been better if the project was to be completed again over a period. This will predominantly draw on the periods of the projects that could have been improved.

Moreover, the project was challenging, one way in which it could have been improved is by managing the time better that was allocated to the completion of it. There was a Gantt chart at the proposal stage, but unpredictable variables were not factored into this Gantt chart, this meant that some chapters were completed easier than others. This result in high levels of pressure to complete the project was being applied. This would be changed if the project could be completed again, a new Gantt chart would be designed, which would factor in time for unpredictable variables, and this would be followed with a stricter attitude. This would ensure smoother completion of the project. Another variable which would be changed if the project was to be done again is the workload the student had at their part-time job, this left them tired and with little time at certain stages of the master's dissertation. If the project was started again, there would be less frequent shifts at their part-time job, so they had more time to optimally focus on their project.

Furthermore, the project focused on a few different areas of software development, predominantly in the field of User Experience, although research was done to increase the existing knowledge the student had. It would have been advantageous if the student had more knowledge in their chosen field of research. This would be changed next time by completing research prior to starting the project so there is a higher level of basic knowledge to begin the project with, this would assist with understanding how to research for academic sources online, this would have also resulted in some chapters being completed in quicker succession, such as the Literature Review.

There were many things learnt from this experience, due to the nature of the project, there had to be extra studying to become familiar with new technologies, this meant that a large portion of the project was spent learning, from the following sources [43] [85] [44] [61] [86]. The student familiarised themselves with JavaScript and React.JS for a section of the implementation, this is significant to the student's learning as they aim to work in the web development industry and React.JS is relevant in that sub-field of the industry, as proven by the Stack Overflow Developer Survey [50]. Due to their motivation to be part of the web development, it was a strong chance to use a blend of other tools that would help them improve as a IT professional, these included, the Google Developer Tools [17], and NPM [66], and design tools for UI design, such as MockUp [54]. These extra tools and technologies will overall contribute to the student's ability to work in the industry alongside experienced professionals. They also deepened their knowledge in the field of UX, which will enable them to contribute to this during their future and through researching a sub-field

such as accessibility, it has influenced them in building a higher level of empathy and consciousness to all potential users of a system. This experience has been challenging overall, it has challenged the student's skill to independently learn, develop, and research.

This section has reflected on the project overall and how it has taught the student to be more resilient, considerate, and understanding towards the architecture of a system and the overall user experience that it intends to convey.

8. Appendices

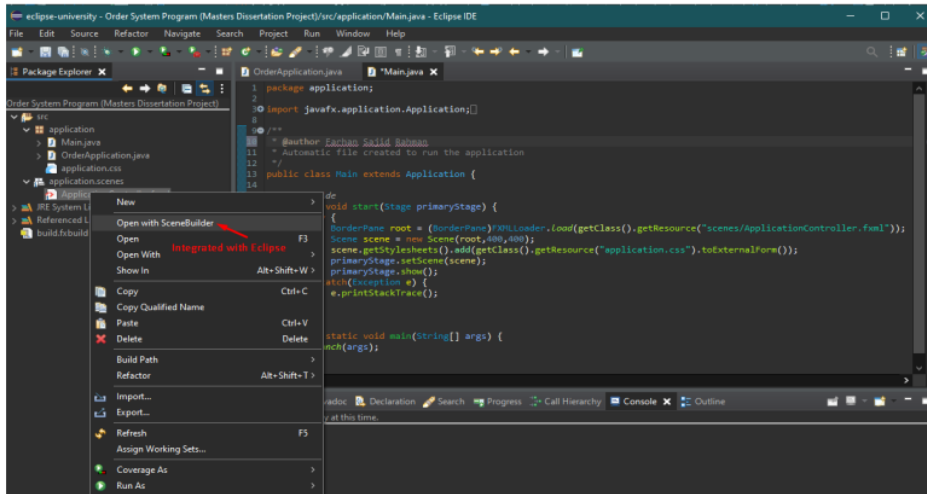
This section includes additional information of the project.

8.1 Appendix A – Design

Showcasing the design procedure of the program that is being built.

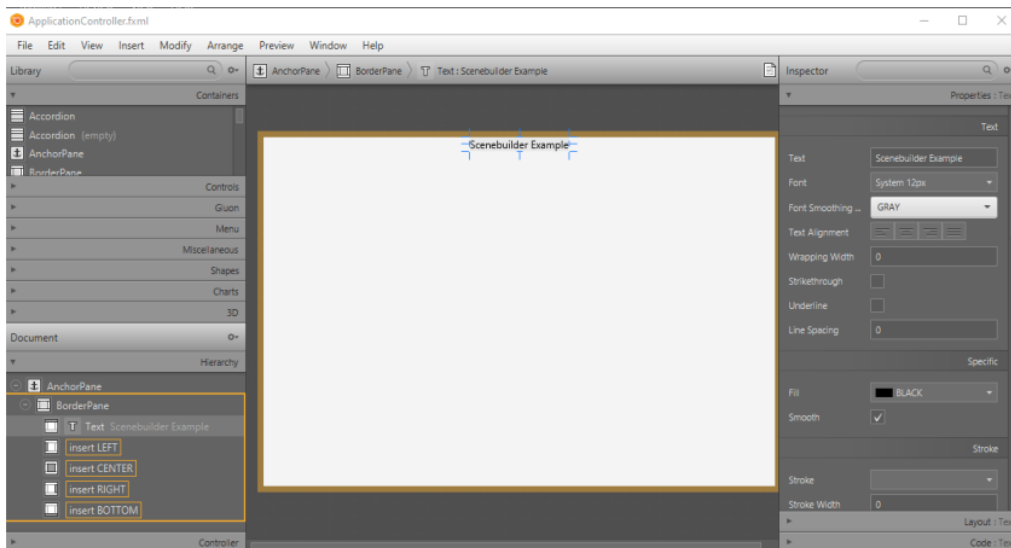
8.1.1 Eclipse

Eclipse IDE that will be used to develop the program, SceneBuilder is integrated which brings ease for creating the user interface.



8.1.2 JavaFX SceneBuilder

Showcasing the SceneBuilder tool to generate JavaFX user interfaces. Properties of each component to edit is on the right, the component library is on the left to drag and drop into the hierarchy to display on the main area.



The format of the file that the interfaces save, is an FXML file. FXML is based on XML and is a markup language which was for building object graphs with Java. It is scriptable and became suited for building interfaces a JavaFX application [65].

```

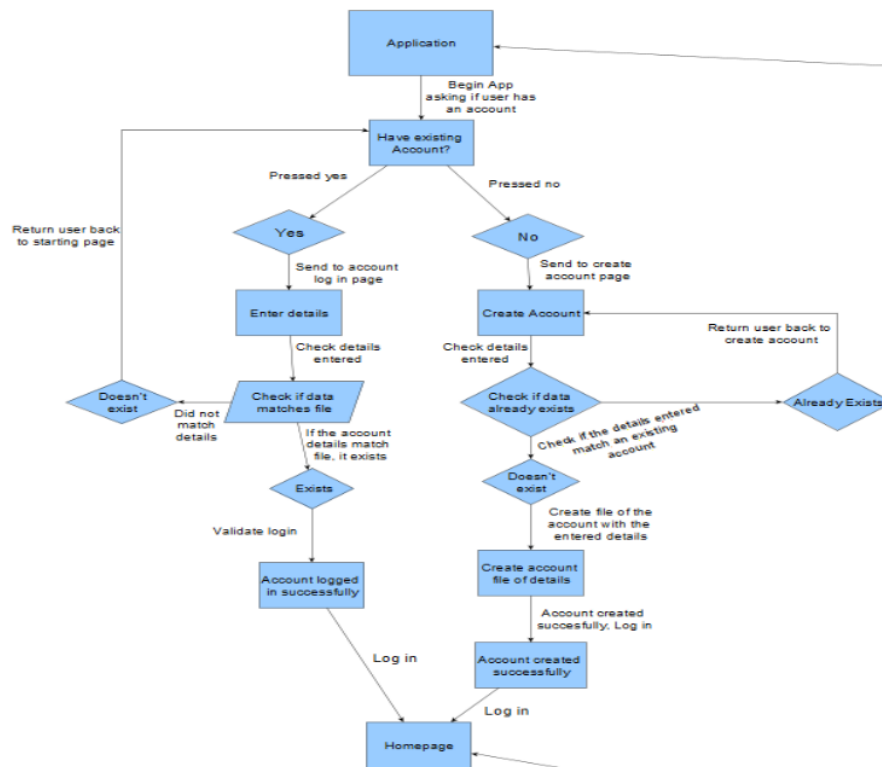
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="556.0" prefWidth="562.0">
  <children>
    <AnchorPane prefHeight="49.0" prefWidth="562.0" AnchorPane.bottomAnchor="480.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="562.0" AnchorPane.topAnchor="0.0">
      <children>
        <Pane prefHeight="82.0" prefWidth="562.0">
          <children>
            <Text layoutX="225.0" layoutY="41.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Order Stuff&#10;Home" />
            <font size="23.0" />
          </children>
        </Pane>
        <Button fx:id="btnBackHome" layoutX="14.0" layoutY="16.0" mnemonicParsing="false" onMouseClicked="#backHome" text="Back to Home" textAlignment="center" />
        <Text layoutX="14.0" layoutY="72.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Back to Home" textAlignment="center" />
      </children>
    </AnchorPane>
    <AnchorPane layoutX="181.0" layoutY="91.0" prefHeight="200.0" prefWidth="200.0" />
    <AnchorPane layoutX="10.0" layoutY="428.0" prefHeight="128.0" prefWidth="562.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="562.0" AnchorPane.topAnchor="428.0">
      <children>
        <Pane prefHeight="128.0" prefWidth="562.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="562.0" AnchorPane.topAnchor="428.0">
          <children>
            <Text layoutX="244.0" layoutY="86.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Checkout" textAlignment="center" />
            <font size="18.0" />
            <Text layoutX="183.0" layoutY="98.0" strokeType="OUTSIDE" strokeWidth="0.0" text="Account &#10;Settings" />
            <font size="18.0" />
          </children>
        </Pane>
      </children>
    </AnchorPane>
  </children>
</AnchorPane>

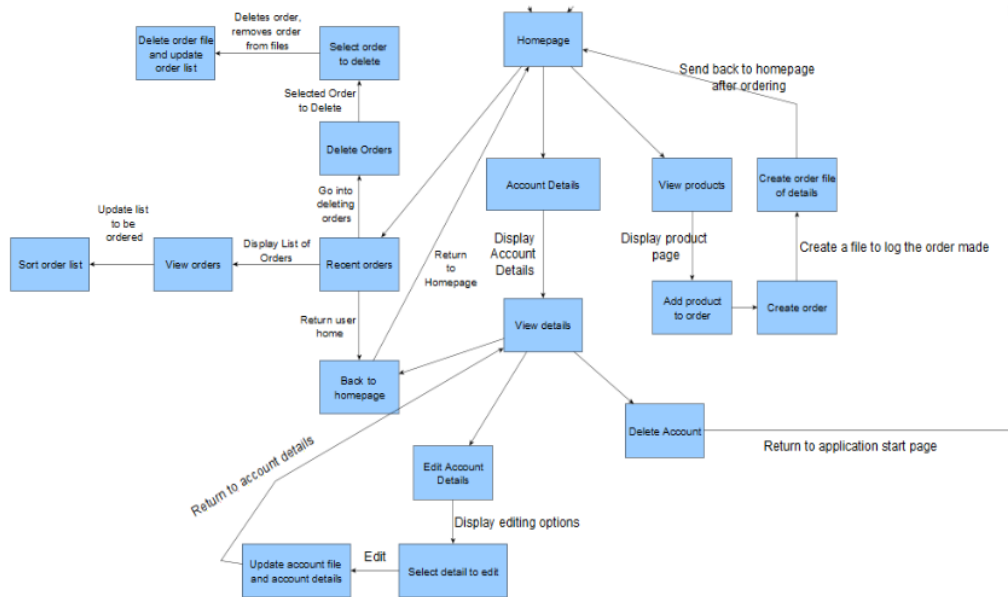
```

8.1.3 Flowchart of the Application

The first image displayed is part one of a flow chart diagram created to visualize how the application will function on the first page on runtime. The second image goes onto part two of the flow chart for displaying how the application will function after the customer logs into their account. It shows what features and capabilities there will be and give idea on how it will work.

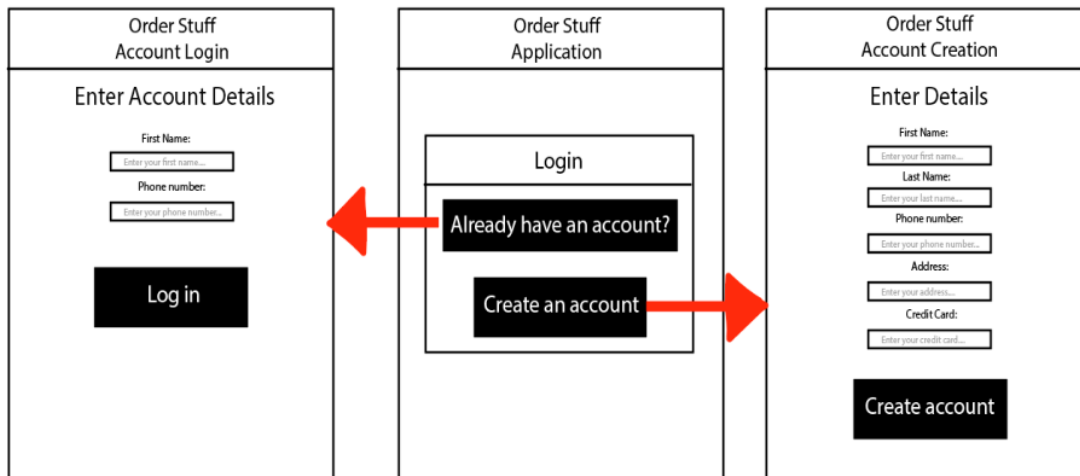


To put it into writing, a customer will log in or create an account and have the ability to navigate from viewing products and creating an order, to viewing and editing their account details, and viewing or deleting any of the orders that were created by them. For every new account that is created, there will be a file output saving the information. This is the same concept to use for orders which are made by the customer too.

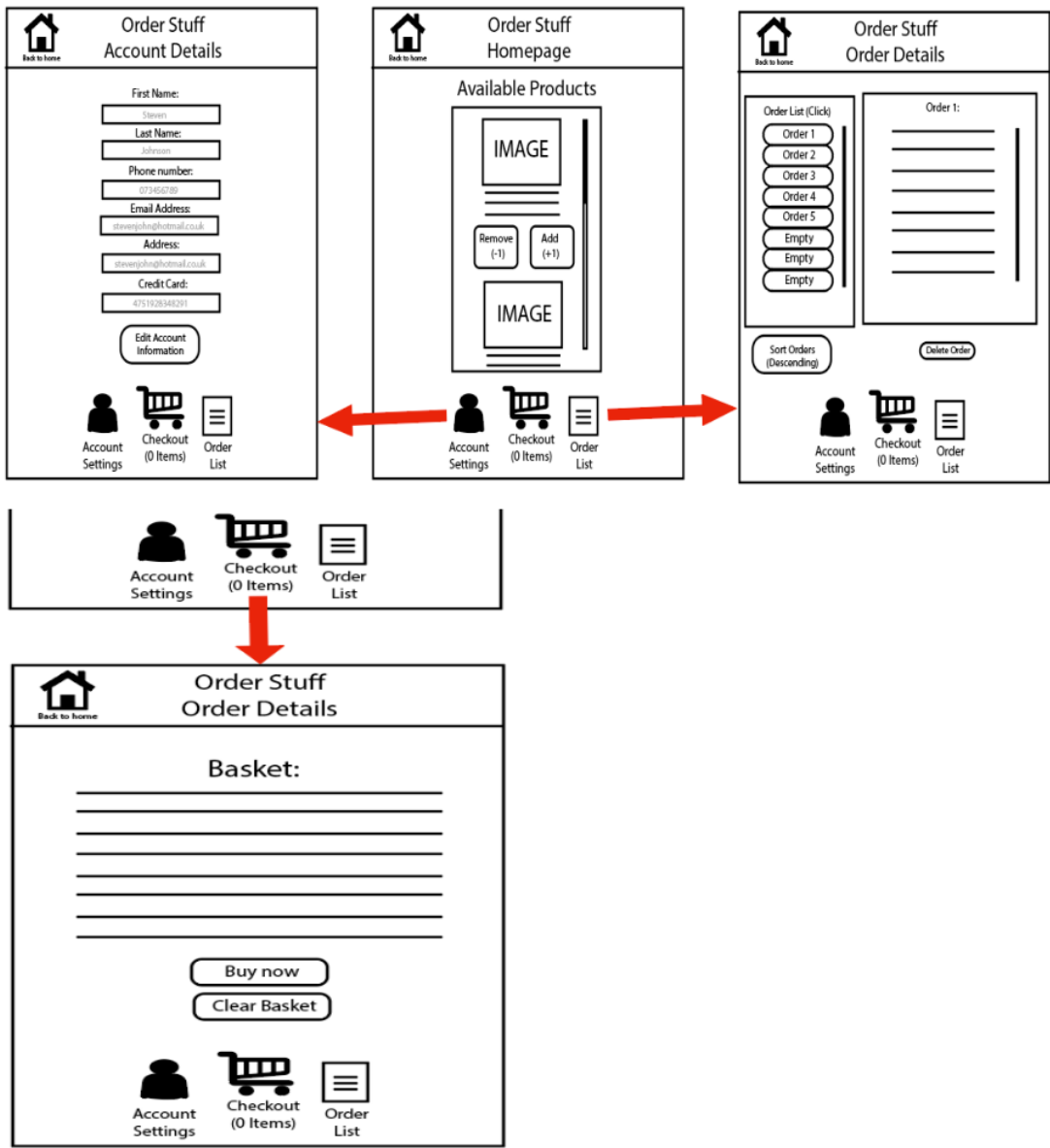


8.1.4 User Interface Storyboards

A storyboard was created to showcase a path for the user interface to be designed like during the implementation stage. There are 7 stories in total and the user interface within the middle with the title of "Order stuff Application" would be the first interface upon loading. The arrows point to the pages the button will navigate to when clicked on.



Once the user has created their account or has logged in, they will be directed to the home page which is displayed on the interface design in the middle, it has the title of “Order Stuff Homepage”. The arrows will take the user to the page of the button that the customer clicks on. On the top left, the back to home icon will work as a button to take the user back to the homepage.

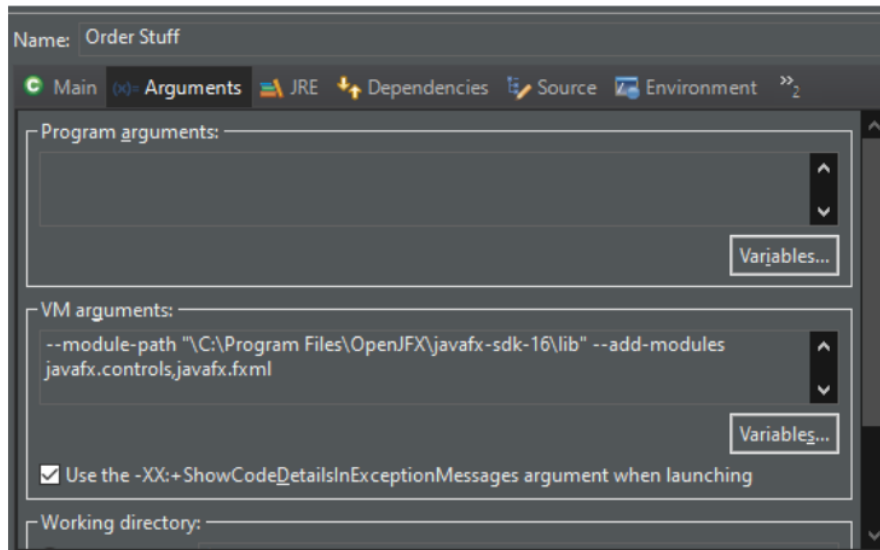


8.2 Appendix B – Implementation Stage

In this appendix, necessary extra information will be given to show parts of the different aspects in the implementation stage of the application.

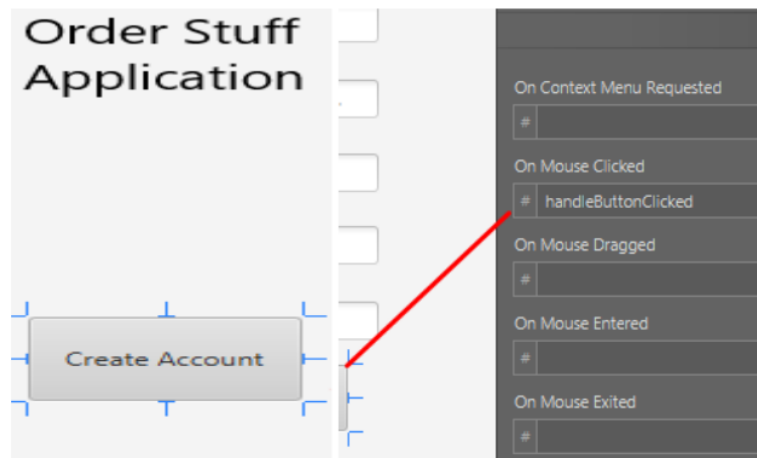
8.2.1 JavaFX Library

Run configuration of the program, the VM arguments were added to link the path of the library and JavaFX FXML module.



8.2.2 User interface component functionality

Once each page of the application is created, the way of adding functionality to the components that will be interacted with such as the buttons, it would be given an id to associate with it. Within SceneBuilder, each component will have properties that can be set. With every component, the code section of the properties allows for setting an id to an action that is wanted. An example that can be given is with a button. In the SceneBuilder tool, the component was selected and then had an id added to “on mouse clicked” for detecting when the button is clicked, and a regular ID can be given that is associated with the button..

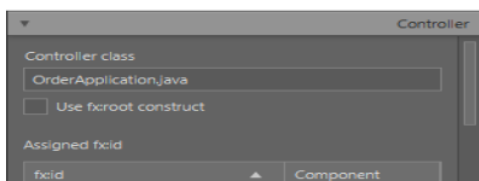


With the id being set, it allows being referenced in code with the JavaFX library imported. It references the FXML file that is output by SceneBuilder with the id given for the button. It is case sensitive and if it matches, the button will perform the code that is written in the body.

```
@FXML
void handleButtonClicked(MouseEvent event) {

}
```

To also mention, each FXML interface that is made with the SceneBuilder tool must set the controller class, this is file where the functionality of components will be read and executed during runtime.



8.2.3 Object Oriented Programming Concepts

This section goes through the concepts that were listed in the design section and displays how it was achieved through the code.

8.2.3.1 Inheritance

The first concept completed was with inheritance. It linked a customer and staff file to the parent class called account which held the linked information both files would require. A pseudocode snippet can be given how it was achieved:

>Account (Parent class): Contains variables and methods that can be made use of by subclasses

>Customer (Sub class of parent):
uses keyword extends to inherit Parent class

>Staff (Sub class of parent):
uses keyword extends to inherit Parent class

Example of code:

```
public class Account {
    // Holds necessary information about an account e.g. First name, Last name
}

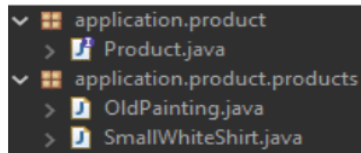
public class Customer extends Account {
    // Inherits and uses the necessary information of an account from the parent
    // Can add extra information in relation to a customer in this class
}

public class Staff extends Account {
    // Inherits and can use the necessary information of an account from the parent
    // Can add extra information in relation to a staff account in this class
}
```

```
}
```

8.2.3.2 Interfaces

After building the customer account files, it went onto identifying where interfaces can be implemented in the code. It was decided to manage products with the use of interfaces, as each product can have the same necessary code which would be the product name and price, and description. This application will also hold the image file link in order to display. Below displays the folder structure, the product interface, and an example of an actual product implementing the interface:



```
package application.product;

import java.io.Serializable;

public interface Product extends Serializable {

    String getProductName();
    double getProductPrice();
    String getProductDescription();
    String getImage();
}
```

```
1 package application.product.products;
2
3 import application.product.Product;
4
5 public class SmallWhiteShirt implements Product {
6
7     private static final long serialVersionUID = -8684035598984334902L;
8
9     @Override
10    public String getProductName() {
11        return "Small White T-Shirt";
12    }
13
14    @Override
15    public String getProductDescription() {
16        return "A small white t-shirt for ages 13+";
17    }
18
19    @Override
20    public double getProductPrice() {
21        return 29.99;
22    }
23
24    @Override
25    public String getImage() {
26        return "/resources/products/SmallWhiteShirt.png";
27    }
28
29    @Override
30    public String toString() {
31        return getProductName();
32    }
33 }
```

9. References

- [1 M. Egele, T. Scholte, E. Kirda and C. Kruegel, "A Survey on Automated Dynamic
] Malware-Analysis," *ACM Computing Surveys*, vol. 44, no. 2, p. 2, 2012.
- [2 B. Johnson, Y. Song, E. Murphy-Hill and R. Bowdidge, "Why Don't Software Developers
] Use Static Analysis Tools to Find Bugs?," *North Carolina State University*, pp. 1-10, 2013.
- [3 R. Bellairs, "What Is Static Analysis? Static Code Analysis Overview," Perforce, 10
] February 2020. [Online]. Available: <https://www.perforce.com/blog/sca/what-static-analysis#static>. [Accessed 25 March 2022].
- [4 A. S. Gillis, "static analysis (static code analysis)," TechTarget, [Online]. Available:
] <https://whatis.techtarget.com/definition/static-analysis-static-code-analysis>. [Accessed 25 March 2022].
- [5 P. Thomson, "Static Analysis: An Introduction," *The fundamental challenge of Software
] Engineering is one of complexity*, vol. 19, no. 4, pp. 1-13, 2021.
- [6 U. Koc, "Improving the usability of static analysis tools using machine learning,"
] *Dissertation, Department of Computer Science*, pp. 24-25, 2019.
- [7 I. Gainulin, "What's the Use of Dynamic Analysis When You Have Static Analysis?,"
] PVS-Studio, 24 July 2019. [Online]. Available: <https://pvs-studio.com/en/blog/posts/0643/>. [Accessed 25 March 2022].
- [8 M. Benson, "Customer Profile NASA Study Explores the Benefits of Static Analysis," p.
] 1.
- [9 M. Striewe and M. Goedicke, "A Review of Static Analysis Approaches for Programming
] Exercises," *International conference on Computer Assisted Assessment*, pp. 1-12, 2014.
- [1 D. Between, "Difference Between Source Code and Bytecode," 2018.
0]
- [1 M. Beller, R. Bholanath, S. McIntosh and A. Zaidman, "Analyzing the State of Static
1] Analysis: A Large-Scale Evaluation in Open Source Software," *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pp. 470-471, 2016 .
- [1 T. Moses, E. Petrovna and M. Barzanti, "Static Analysis: new emerging algorithms," pp.
2] 1-5, 2013.
- [1 P. Li and B. Cui, "A Comparative Study on Software Vulnerability Static Analysis
3] Techniques and Tools," *IEEE International Conference on Information Theory and Information Security*, Vols. 521-524, pp. 1-4, 2010.
- [1 "Software Engineering | Control Flow Graph (CFG)," Geeksforgeeks, 15 May 2019.
4] [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-control-flow-graph-cfg/>. [Accessed 28 August 2022].

- [1 L. Q. D. Nguyen, J. R. Wright and K. Ali, "Why Do Software Developers Use Static
5] Analysis Tools? A User-Centered Study of Developer Needs and Motivations," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, pp. 835-847, 2020.
- [1 A. M. Perhinschi , "Static Code Analysis: On Detection of Security Vulnerabilities and
6] Classification of Warning Messages," *Lane Department of Computer Science and Electrical Engineering*, pp. 1-2, 2015.
- [1 L. Nguyen Quang Do, K. Ali, B. Livshits, E. Bodden, J. Smith and E. Murphy-Hill, "Just-
7] in-Time Static Analysis," *ISSTA 2017: Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, p. 307, 2017.
- [1 "About Objective-C," Developer Apple, [Online]. Available:
8] <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>. [Accessed August 2022].
- [1 "Static Analysis Tool Exposition VI (SATE VI)," NIST, 22 May 2019. [Online]. Available:
9] <https://samate.nist.gov/SATE.html>. [Accessed 25 March 2022].
- [2 "Source Code Security Analyzers," NIST, 25 March 2022. [Online]. Available:
0] <https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers>. [Accessed 23 March 2021].
- [2 D. Hovemeyer and W. Pugh, "Finding Bugs," *Dept. of Computer Science, University of Maryland*, pp. 1-14, 2004.
- [2 D. Singh, V. Ramachandra Sekar, K. T and B. Johnson, "Evaluating How Static Analysis
2] Tools Can Reduce Code Review Effort," *Conference: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1-2, 2017.
- [2 S. Wagner, J. Jurjens, C. Koller and P. Trischberger, "Comparing Bug Finding Tools with
3] Reviews and Tests," *International conference on Testing of Communicating Systems*, pp. 40-55, 2005.
- [2 "FindBugs™ Fact Sheet," FindBugs, [Online]. Available:
4] <http://findbugs.sourceforge.net/factSheet.html>. [Accessed August 2022].
- [2 "Commons BCEL," Apache Commons, [Online]. Available:
5] <https://commons.apache.org/proper/commons-bcel/>. [Accessed 20 August 2022].
- [2 "Checkstyle," Checkstyle, [Online]. Available: <https://checkstyle.sourceforge.io/>.
6] [Accessed 2022 August 2022].
- [2 "Google Java Style Guide," Checkstyle, [Online]. Available:
7] <https://checkstyle.sourceforge.io/styleguides/google-java-style-20180523/javaguide.html#s6-programming-practices>. [Accessed 28 August 2022].
- [2 PRQA, "Succeeding with static code analysis: an implementation guide white paper,"
8] *Avoiding pitfalls by following best practice planning*, p. 5, 2016.
- [2 MISRA, "MISRA Compliance: 2020 - Achieving compliance with MISRA Coding
9] Guidelines," 2020.

- [3 D. K. Oka, "Static Code Analysis for Automotive Software," *Building Secure Cars: Assuring the Automotive Software Development Lifecycle*, pp. 63-90, 2021.
- [3 R. N. Charette, "Why Software Fails," p. 47, 2005.
- 1]
- [3 D. Burke, "All Circuits are Busy Now: The 1990 AT&T Long Distance Network Collapse," 2] *California Polytechnic State University*, p. 1, 1995.
- [3 A. Gosain and G. Sharma, "Static Analysis: A Survey of Techniques and Tools," 3] *Advances in Intelligent Systems and Computing*, pp. 581-589, 2015.
- [3 N. Nagappan and T. Ball, "Static analysis tools as early indicators of pre-release defect 4] density," *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pp. 580-586, 2005.
- [3 "Overflow and Underflow of Buffers," Apple Developer Documentation, [Online]. 5] Available: <https://developer.apple.com/documentation/xcode/overflow-and-underflow-of-buffers>. [Accessed 18 August 2022].
- [3 "Buffer underflow," TechTarget, [Online]. Available: 6] <https://www.techtarget.com/searchsecurity/definition/buffer-underflow>. [Accessed 18 August 2022].
- [3 L. Sakerman and R. Hallikas, "Overview of the advantages and disadvantages of static 7] code analysis tools," *Institute of Computer Science*, pp. 1-4, 2021.
- [3 U. Koc, S. Wei, J. S. Foster, M. Carpuat and A. A. Porter, "An Empirical Assessment of 8] Machine Learning Approaches for Triaging Reports of a Java Static Analysis Tool," *Department of Computer Science*, pp. 1-2, 2019.
- [3 P. Emanuelsson and U. Nilsson, "A Comparative Study of Industrial Static Analysis 9] Tools," *Electronic Notes in Theoretical Computer Science*, p. 5–21, 2008.
- [4 H. Tahbaldar, P. Borbora and G. P. Khataniar, "Teaching automated test data generation 0] tools for C, C++ , and Java programs," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 5, no. 1, pp. 1-3, 2013.
- [4 M. Nachtigall, L. N. Quang Do and E. Bodden, "Explaining Static Analysis – A 1] Perspective," *Conference*, pp. 29-32, 2019.
- [4 B. Cornelissen, A. Zaidman, A. v. Deursen, L. Moonen and R. Koschke, "A Systematic 2] Survey of Program Comprehension through Dynamic Analysis," *IEEE transactions on Software Engineering*, vol. 0, no. 0, pp. 1-4, 2000.
- [4 I. Gomes, P. Morgado, T. Gomes and R. Moreira, "An overview on the Static Code 3] Analysis approach in Software Development," *Software Testing and Quality, Master in Informatics and Computing Engineering*, pp. 1-13, 2009.
- [4 B. Chess and G. McGraw, "Static Analysis for Security," *IEEE Security and Privacy*, vol. 4] 2, no. 6, pp. 76-78, 2004.

- [4 A. Chandrababu, "A Comparison between Agile and Traditional Software Development Methodologies," *The University of Western Australia*, pp. 11-12, 2005.
- [4 F. Anwar, S. Aftab, U. W. waheed and S. M. Shah, "Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey," *International journal of multidisciplinary sciences and engineering*, pp. 1-5, 2017.
- [4 "Eclipse IDE for Java Developers," Eclipse Foundation, [Online]. Available:
7] <https://www.eclipse.org/downloads/packages/release/2022-06/r/eclipse-ide-java-developers>. [Accessed 22 August 2022].
- [4 "Notes on the Eclipse Plug-in Architecture," Eclipse, [Online]. Available:
8] https://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html. [Accessed 22 August 2022].
- [4 "FindBugs Eclipse Plugin," Eclipse Marketplace, [Online]. Available:
9] <https://marketplace.eclipse.org/content/findbugs-eclipse-plugin>. [Accessed 22 August 2022].
- [5 "pmd-eclipse-plugin," Eclipse Marketplace, [Online]. Available:
0] <https://marketplace.eclipse.org/content/pmd-eclipse-plugin>. [Accessed 22 August 2022].
- [5 "JavaFX Scene Builder: Using JavaFX Scene Builder with Java IDEs," Oracle, [Online].
1] Available: <https://docs.oracle.com/javase/8/scene-builder-2/work-with-java-ides/sb-with-eclipse.htm#JSBID104>. [Accessed 22 August 2022].
- [5 "JavaFX: Getting Started with JavaFX," Oracle, [Online]. Available:
2] <https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm>. [Accessed 22 August 2022].
- [5 "JavaFX Scene Builder," Oracle, [Online]. Available:
3] <https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html>. [Accessed 22 August 2022].
- [5 "What Is Inheritance?," Oracle, [Online]. Available:
4] <https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html>. [Accessed 22 August 2022].
- [5 "What Is an Interface?," Oracle, [Online]. Available:
5] <https://docs.oracle.com/javase/tutorial/java/concepts/interface.html>. [Accessed 22 August 2022].
- [5 "Abstract Methods and Classes," Oracle, [Online]. Available:
6] <https://docs.oracle.com/javase/tutorial/java/landl/abstract.html>. [Accessed 22 August 2022].
- [5 "Polymorphism," Oracle, [Online]. Available:
7] <https://docs.oracle.com/javase/tutorial/java/landl/polymorphism.html>. [Accessed 1 August 2022].

- [5] "Database Java Developer's Guide," Oracle, [Online]. Available:
8] https://docs.oracle.com/cd/B19306_01/java.102/b14187/chone.htm#BABHGCBA.
[Accessed 22 August 2022].
- [5] D. Smith, "Java Client Roadmap Updates," Oracle Blogs, 13 May 2020. [Online].
9] Available: <https://www.oracle.com/java/technologies/java-se-support-roadmap.html>.
[Accessed 28 August 2022].
- [6] "JDK 19," OpenJDK, 21 July 2022. [Online]. Available:
0] <https://openjdk.org/projects/jdk/19/>. [Accessed 28 August 2022].
- [6] D. Smith, "The Future of JavaFX and Other Java Client Roadmap Updates," Oracle
1] Blogs, 7 March 2018. [Online]. Available: <https://blogs.oracle.com/java/post/the-future-of-javafx-and-other-java-client-roadmap-updates>. [Accessed 28 August 2022].
- [6] "JavaFX," OpenJFX, [Online]. Available: <https://openjfx.io/>. [Accessed 28 August 2022].
2]
- [6] "Introduction," SpotBugs, [Online]. Available:
3] <https://spotbugs.readthedocs.io/en/stable/introduction.html>. [Accessed 1 September 2022].
- [6] "Using the SpotBugs Eclipse plugin," Spotbugs, [Online]. Available:
4] <https://spotbugs.readthedocs.io/en/stable/eclipse.html>. [Accessed 1 September 2022].
- [6] "Introduction to FXML," Oracle, 2012. [Online]. Available:
5] https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html.
[Accessed 22 August 2021].

Farhan Sajid Rahman Masters project

GRADEMARK REPORT

FINAL GRADE

/100

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

