Visualizing Security Vulnerability Evolution of Software Systems

N.N. Sinhabahu

2020

Visualizing Security Vulnerability Evolution of Software Systems

A dissertation submitted for the Degree of Master of Science in Computer Science

N.N. Sinhabahu
University of Colombo School of Computing
2020

DECLARATION

I hereby declare that the thesis is my original work, and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Student Name:N.N Sinhabahu

Registration Number:2017/MCS/077

Index Number:17440771

_____

Signature of the Student & Date

This is to certify that this thesis is based on the work of Mr. /Ms. _____ under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr G.D.S.P.Wimalaratne

_____

Signature of the Supervisor & Date

Abstract

The analysis of large-scale software and finding security vulnerabilities while its evolving is difficult without using supplementary tools, because of the size and complexity of today's systems. However, just looking at a report may not transmit the overall picture of the system in terms of security vulnerabilities and its evolution throughout the project lifecycle. Software visualization is a program comprehension technique used in the context of the present and explores large amounts of information precisely. For the analysis of security vulnerabilities of complex software systems, Secure Codecity with evolution is an interactive 3D visualization tool that can be utilized. It utilize techniques and methods that are used in graphical visualualization to illustrating security aspects and the evolution of software. The main goal of the proposed framework defined as uplift, simplify, and clarify the mental representation that a software engineer has a software system and its evolution in terms of its security. Static code was visualized based on a city metaphor, which represents classes as buildings and packages as districts of a city. Identified Vulnerabilities were represented in a different color according to the severity. To visualize different aspects, large variety of options were given. Users can evaluate the evolution of the security vulnerabilities of a system on several versions using matrices provided which will help users go get an overall understanding of security vulnerabilities varies with different versions of the software. This framework was implemented using SonarQube for software vulnerability detection and ThreeJs for implementing the City Metaphor. The evaluation results evidently show that our framework surpasses the existing tools in terms of accuracy, efficiency, and usability.

Keywords : 3D software visualization, Vulnerability Evolution, Re-engineering ,Vulnerability Analysis, 3D graphics, human-computer interaction.

Acknowledgment

First, I would like to express my sincere gratitude to Prof K.P Hewagamage, Director, University of Colombo, School of Computing for giving me the opportunity to carry out this research study and for their supervision given throughout this study.

I wish to express my profound gratitude to my Supervisor Dr.P.Wimalarathne, Senior lecturer, Department of University of Colombo for his excellent supervision and guidance.Mr.Chaman Wijesiriwardena,Lecturer faculty of information technology Moratuwa for the guidance,All the students who's involved in "Secure Codecity" Research Project which is use as the basis of this research.

My sincere thanks goes to all academic staff of University of Colombo who donated their valuable time for the success of my research study.

Finally, I express my gratitude to my family & all those colleagues that are numerous to name & wishes to remain anonymous, for their generous assistance to make this research a success.

Nadun Sinhabahu

Table of Contents

Table of Figures

List of Tables (LOT)

Chapter 1: Introduction

1.1 Statement of the problem

The software industry has grown dramatically in recent decades. Since then the complexity of the codes has grown up. Application developers write these features, rely on their operation, and may even re-use them in their code. Due to rapid, feature-driven development and code sharing, when a vulnerability is introduced in code (and goes undetected) it can spread rapidly.

With the advancement in the software industry, a new software product is out to the market every passing day. But very few are having the expected security standards and follow Information Security principles. Applications may simply use the same technology stack and have no common business function in an organization. Unfortunately, code reviews are done by only some of the developers. Most of them ignore it up until the main security breach happens.

Some of these vulnerabilities may be identified during the testing process. Most of these go undetected until some security breaches happen. Due to growing competition, project delivery time is shortened and the security factor is compromised to make systems more up to date, which will eventually lead to unauthorized access and data/information theft.

Hence there is a need for a proper mechanism to evaluate code and identify the potential vulnerabilities in the software product from the early stages of the project and keep the track of them in the entire project time span. If the solution can keep the track of the security-related issues which have been resolved in the current stage, will encourage developers to take precautions to prevent the security loopholes.

1.2 Motivation

Nowadays, Software Security plays a key role in the Application development process with complex requirements in hand. This makes applications with thousands and millions of lines. Which leads to several kinds of vulnerabilities in software products. Since the size and the complexity of the products, the manual code review is not an option anymore[4].

There are existing applications to fulfil this task[5], But using only these, users can’t get an overall idea about the security level of the project. These tools do not cater to the need for a simpler and user-friendly way of understanding the security vulnerabilities of a project.

There are code visualization techniques available. But there is a problem exists such as how to utilize the visualization techniques to visualize the identified vulnerabilities in a way where users can understand properly.

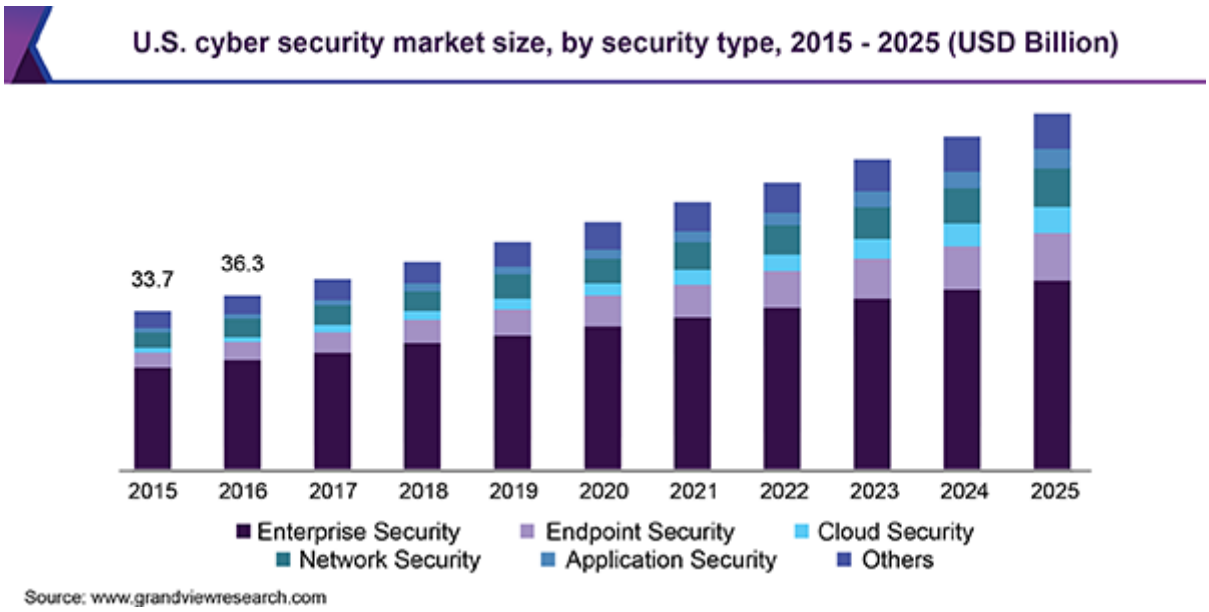


Figure 1.1Cost Related to security over the time[36]

1.3 Research significance and Previous work

Most of the researches have been conducted on finding vulnerabilities of a software project without executing it through static code analysis . Only a few researches were there addressing the problem of a lack of understanding about the security aspects of a software project. Another problem was that it's difficult to find research addressing both of these problems together. The literature review-based studies focus on the analysis of previously conducted studies and other papers. Existing approaches have not been focused on finding the vulnerabilities of software products but existing systems unable to communicate it to the people. Hence it's difficult to evaluate the software in terms of security. Therefore considering the above facts, there is a need for a comprehensive study regarding this research topic.

Previously research has been conducted to address this issue by UCSC students("Secure Codecity" Research Project)[2], and this research is an extension of that research.

1.4 Aims and Objectives

1.4.1 Aim

Provide a mechanism to give users a better understanding of the security aspects of software solutions by finding and visualizing vulnerabilities that exist in source code in different versions via using static code analysis and 3D visualization techniques alongside with related countermeasures in order to enhance the secureness of the a software in Software Development Life Cycle.

1.4.2 Objectives

- To Perform a background on software security vulnerabilities and what's causing of them
- To study about existing vulnerability discovering process
- To compare existing software visualization metaphors
- To select a suitable visualization metaphor which can be used in the proposed solution
- To study existing static code analysers available
- To select a suitable static code analyser for the proposed solution

- To Implement the proposed Solution
- To Evaluate the proposed solution using benchmarking projects

1.4 Research questions and their Objectives

As previously mentioned, this study aimed to improve the understandability of the vulnerability aspects of software products by using visualization techniques. In order to achieve these objectives, following research questions (RQs) were proposed.

RQ1: How to extend code city metaphor to visualize source code and its evolution by referring to the top 10 security vulnerabilities identified by The Open Web Application Security Project (OWASP)? [6],[32]

RQ2: How to improve the understandability of the vulnerability aspects by visualizing the vulnerabilities over the different versions of software? [32]

Throughout this study above research questions have been answered. This study consists of a literature review that provides past important points related to the research topic and research questions.

Table 1.1 shows the research questions that this study is focused on.

No	Research question	Objectives
RQ1	How to extend code city metaphor to visualize source code evolution by referring to the top 10 security vulnerabilities?	This research question's objective is to give identify top10 security vulnerabilities by static code analyzing techniques and visualize these identified technologies via code city metaphor with combining the visualization techniques for better user understandability
RQ2	How to improve the understandability of the vulnerability aspects by visualizing the vulnerabilities over the different versions of software?	This research question's objective is to explore how to improve user understanding about the vulnerability changes with the code evolution of software by using visualization techniques.

Table 1.1 Research questions and their Objectives

1.5 Scope

Secure Code city with Evolution is capable of analyse the source code in different versions of the java web-based applications and visualizing its security related information using code city metaphor. Due to using SonarQube for the vulnerability analysing part and SonarQube identifying Java application project vulnerabilities with a higher accuracy compared with other languages solution limited only for the java web-based applications. The system categorizes identified

vulnerabilities into categories according to their severity. There is a colour code for each category which represents the color of the buildings in the modelled city

Framework only focusing on identifying the OWASP Top10 listed vulnerabilities. Other vulnerabilities identified by the SonarQube wont be processed by the framework.This framework will keep the track of vulnerabilities identified and resolved in each version used these processed information will be used for visualizing vulnerability evolution between two different versions. . By comparing different versions of the system, users can identify how the security vulnerabilities evolve with new feature additions and changes. Users can visualize the security evolution of a system using different models given matrices.

1.6 Structure of the Thesis

The structure of the thesis is as follows. Discuss the background and related work(Literature review), Research Methodology, Discuss the Proposed solution in detail, Present and discuss result from the study, conclusion and future work

Chapter 2: Literature Review

2.1 Introduction

In Recent years cybersecurity crimes grew up and security has become a major concern. Security-related crimes have gone up and associated cost increased accordingly [10]

Since the complexity of applications, grown-up security vulnerabilities in such applications have grown as well. There are different kinds of application vulnerabilities. In order to find a better solution studying the existing Web Application Security Mechanisms is a must [1]

Information security aspects have changed from time to time due to technology and market changes. For example, over the past 10 years, people have shown concerns about cloud computing or privacy or third-party public clouds, whereas today using a cloud service is much more widely accepted because cloud providers have more security readiness compared to the past, and business sectors are satisfied with the benefits of the cloud, for example, low cost and flexibility[14].

Bug prediction is one of the most active research areas in software engineering and different prediction techniques have been proposed by the research community. This chapter describes major approaches in software defect prediction.

2.2 Problem Definition

Security vulnerabilities in software systems have posed a serious threat to users, organizations and even nations. In 2017, unpatched vulnerabilities allowed the WannaCry ransomware crypto worm to shut down more than 300,000 computers around the globe[15]. At the same time, another vulnerability in Equifax's Apache servers led to a devastating data breach that exposed half of the American population's Social Security Numbers [16]. As of December 2017, the CVE website has archived more than 95,000 security vulnerabilities.

By Analysing these statistics it's clear that software security breaches have increased dramatically in recent years, the costs associated with them increases accordingly. it's clear security has become a major concern for software products more than ever.

In order to resolve these vulnerabilities, real cause of the issues should be identified. It's important to augmenting the concerns of software security into each phase of the Software development Life Cycle(SDLC) and keep the track of the vulnerabilities detected and vulnerabilities resolved between different versions.

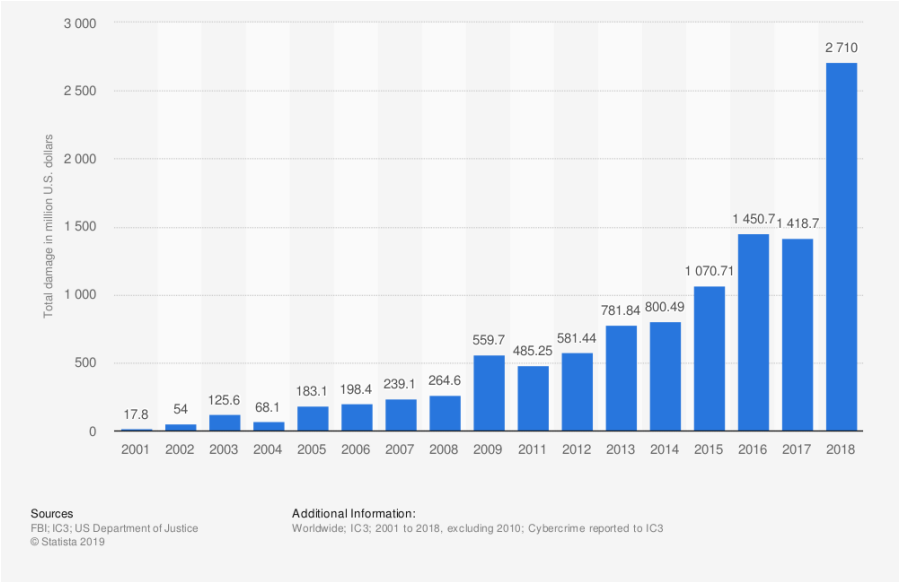


Figure 2.1. Amount of Monetary Damages caused by cybercrimes over the years[37]

2.3 Background Analysis

In the early days it was believed that the complexity of software would lead to defects and security threats.To show how complicated the software is, Akiyama built a simple model using Lines of Codes(LOC) [17]. Using LOC as a metric for vulnerability assessment was too simple and therefore MaCabe proposed cyclomatic complexity as a measure for security vulnerability prediction[18]. Cyclomatic complexity and Halstead complexity [19] were very popular metrics for evaluating vulnerabilities at that time but there was a major drawback in those models. The model can be used to predict on the new software module and so they have demonstrated some relationship between the matrix and the number of errors[20] Shen et al built a linear regression model in order to test the accuracy of the defects identified in the new software module. However, there are some bias issues in that model and Munson et al.proposed a classification model which was modified and had high accuracy [21]. With the increasing popularity of version control systems, several process matrix estimation models were proposed in the 2000s. There were certain limitations in vulnerability prediction models developed during the 2000s.Major limitation was the inability to predict defects whenever a source code file is changed. Just In Time(JIT) security vulnerabilities prediction models were introduced to overcome this limitation and it is also an active research area that allows predicting defects whenever we change the source code. Another drawback is the failure to evaluate new projects and projects with very little historical information. Cross defect prediction models have been introduced to address this limitation.

2.3.1 Evaluating Security Vulnerabilities in a Project

Many Security problems are caused by bugs that can be spotted in the code Ex: Miss using various string functions. Developers ignore the vulnerabilities until problems occur [35].

In the process of identifying Software Vulnerabilities, can be classified into two categories. software vulnerability analysis and software vulnerability discovery. Software vulnerability analysis is mainly focused on analyzing discovered software vulnerabilities to identify the characteristics of vulnerabilities, such as main cause, position and implement features, and characteristics of vulnerability the discovery process, such as features of vulnerability discovery rate.[3]

2.3.2 Common Security vulnerabilities

Web and mobile applications are facing various attacks each and every day. When considering the top critical web application vulnerabilities, poor programming approach which leads to these vulnerabilities [34] which makes the developers responsible for these vulnerabilities. There are various web and mobile applications related vulnerabilities that exist in the present. Also new vulnerabilities are discovered by attackers very frequently. New technologies like cloud infrastructure, new programming languages change the threat landscape and create new attack vectors. This situation makes security more complicated and bizarre for the organizations and makes it easier to the attackers. Since the situation is getting worse day by day, it would be nice to have an independent body or organization who can invest in researching new threats, vulnerabilities, define the severity of the vulnerabilities and define guidelines and best practices to avoid, address these vulnerabilities. Also, they can suggest required and best security solutions, providers and necessary tools. Then the organizations can get a clear idea about the top vulnerabilities and take necessary actions like, educate the engineers, focus on test cases to cover necessary scenarios. This will be a great advantage since it can save considerable resources for an organization. Couple of well-known independent foundations or organizations exist, performing security related research and doing great help for businesses as well as the community. Below are some of them

- Open Web Application Project (OWASP)
- Cigital
- SANS

The following can be identified as the most common Security Vulnerabilities according to OWASP

SQL Injection, Broken Authentication, Sensitive data exposure, XML External Entities (XXE), Broken Access control, Security misconfigurations, Cross-Site Scripting (XSS), Insecure Deserialization, Using Components with known vulnerabilities, Insufficient logging, and monitoring [26]

1	Verbose server banner	8%
2	Weak SSL ciphers	6%
3	Hidden directory detected	6%
4	Clickjacking (aka UI Redressing)	5%
5	Weak password policy	5%
6	Secure cookie attribute not set	5%
7	Cacheable SSL pages	4%
8	SSL/TLS beast information leakage	4%
9	Username enumeration through password reset	3%
10	Reflected cross-site scripting (XSS)	3%
11	HttpOnly cookie attribute not set	3%
12	Verbose error messages	2%
13	Unencrypted viewstate	2%
14	Cross-site request forgery (CSRF)	2%
15	TLS/SSL not enforced	2%
16	Sensitive information leaked via query string parameter	2%
17	TLS/SSL not enabled	2%
18	Application error	2%
19	No account lockout policy	2%
20	Session identifier set prior to authentication	2%

Cigital

Figure 2.2Top 20 Vulnerabilities by CIGITAL[38]

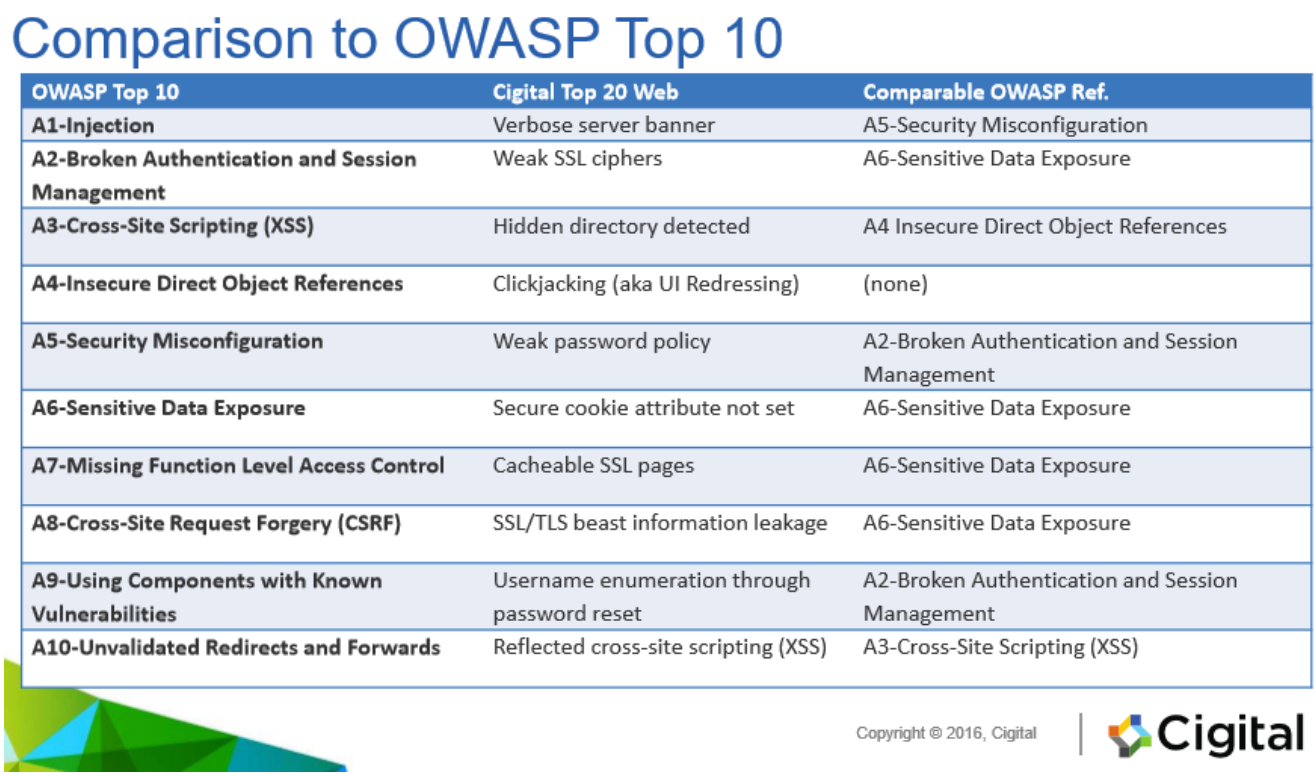


Figure 2.3Comparison to OWASP Top 10

CWE/SANS TOP 25 Most Dangerous Software Errors

Following contains Top 25 Software Errors identified by Common Weakness Enumeration(CWE).This a list which demonstrates most common and critical weaknesses which may lead to software vulnerabilities.These vulnerabilities can be discovered easily and exploit them.

Improper Restriction of Operations within the Bounds of a Memory Buffer, Improper Neutralization of Input during Web Page Generation ('Cross-site Scripting'),Improper Input Validation, Information Exposure, Out-of-bounds Read, Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'),Use After Free, Integer Overflow or Wrap around, Cross-Site Request Forgery (CSRF),Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'),Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'),Out-of-bounds Write, Improper Authentication, NULL Pointer Dereference, Incorrect Permission Assignment for Critical Resource, Unrestricted Upload of File with Dangerous Type, Improper Restriction of XML External Entity Reference, Improper Control of Generation of Code ('Code Injection'),Use of Hard-coded Credentials, Uncontrolled Resource Consumption, Missing Release of Resource after Effective Lifetime, Untrusted Search Path, Deserialization of Untrusted Data, Improper Privilege Management, Improper Certificate Validation

2.3.3 Problems associated with Manual Code Reviewing

Peer code review is a well-established practice among development teams aiming to produce high-quality software, in both open source and commercial environments. According to the statistics, its clear that the formal code inspections will improve the quality of software delivered significantly[13].

In order to conduct a static code analysis, these known characteristics of vulnerabilities should be identified. Since manual code reviews are time-consuming, costly and error-prone, the need for automated solutions has become obvious. Static analysis is the process of evaluating a system or component based on its form, structure, content, or documentation, which does not require program execution[4]

2.4 Mapping study

2.4.1 Available Static Code Analysis Techniques

In order to choose a suitable static code analyser, it should have to be analysed the characteristics, capabilities, and problems in current static code analyser techniques. There are existing source code analysers. Source code analysis tools, also referred to as Static Application Security Testing (SAST) tools, which was designed to analyse source code and/or compiled versions of code to help find security flaws.

Most of the current static code analysers available right now are used as flow-sensitive, interprocedurally and context-sensitive data flow analysis to discover vulnerable points in a program[12]

There are some tools that are used lexical analyser techniques to find vulnerabilities in the source code Ex: ITS4, FlawFinder, and RATS. In these systems, the technique of tokenizing source files is used and then they are matched with the resulting token stream against a library of vulnerable constructs.[5]

There are many static code analysers available. These tools have their strengths, weaknesses, and performance characteristics. While using multiple static code analysers, tools claim to check the same vulnerabilities but generate different results. In this scenario, at least one of the SCA tools is generated with both false positives, which are locations in source code that are incorrectly labelled to have a flaw, and false negatives, which are locations in source code that actually have a flaw and are not labelled at all. Hence it is needed to identify the best-suited static code analyser to suit our purpose. In order to do that, Software Engineering Metrics to Evaluate the Quality of Static Code Analysis Tools can be used[11]

OWASP LAPSE

Advantages	Disadvantages
Possible to integrate with an integrated development environment and perform the source validation without compilation.	Only support for eclipse integrated development environment
Tool handles the testing with three steps, which are identifying the vulnerability source in the source code, identifying the vulnerability sink in	No new versions after 2012

the tool and examine to see whether we can use vulnerability sink to each the vulnerability source	

Table 2.1 Advantages and Disadvantages OWASP LAPSE

YASCA

Advantages	Disadvantages
Possible to integrate with other powerful and related tools	Capable only for finding straight forward, low-hanging fruits and Cross-Site scripting and SQL injections attacks

Table 2.2 Advantages and Disadvantages YASCA

SONARQUBE

Advantages	Disadvantages
Open Source Report generation and the ease of integrating it with Automation servers like Jenkins Code analyzing for detecting vulnerabilities with higher accuracy Multiple Language Support	Accuracy of vulnerability detection well performed only on JAVA code based systems

Table 2.3 Advantages and Disadvantages YASCA

2.4.2 Visualization for complex systems

"Visualization is a method of computing. It transforms the symbolic into the geometric, enabling researchers to observe their simulations and computations. Visualization offers a method for seeing the unseen. It enriches the process of scientific discovery and fosters profound and unexpected insights. In many fields, it is already revolutionizing the way scientists do science." [23]

In order to debug and understand the software systems, diagrams are drawn to visualize what is happening. These diagrams, visualizations will formulate the way that our imagination about software. But there are many problems associated with these techniques. Thinking ahead of time, that most programmers don't have the graphics or compute hardware needed to take advantage of visualizations that have been produced. Ex: Thinking ahead of time Fail to communicate the idea to the users

By considering all these factors there is a clear need for software visualization. The aim of the research is to change the focus of our software visualization efforts which will make sure developers are in touch with reality.

Software visualization is a technique that can be used to summarize the system which can be useful to software maintenance, reverse engineering, and software evolution analysis. After combining knowledge gathered in security and vulnerability analysis with the virtualization, users can detect and take precautions to avoid potential security breaches[9]

2.4.3 CodeCity Metaphor for the visualization

Software is virtual and also it's intangible [24]. Without a visualization technique, it's very hard to make a clear mental representation of what a piece of software it is. Basically, visualizing software is like drawing a picture of the software [25].

Researchers have proposed many software visualization techniques and various taxonomies [27], [28], [29], [30]. There have been many Programs developed to visualize the static code

Ex: Imagix 4D, NDepend, Sotoarc, Sourcetrail, Softagram, Getaviz, SonarGraph[31]

Many of the existing solutions have failed to communicate relevant information about the system to its users. As for the researchers identified, this happens mainly because most of the tools using additional 3rd dimensions to communicate the information to the users, which will lead to information overload. Software's are mostly represented as nodes and Edges in a 3D space. This research has suggested a new approach, which is going to visualize the source code and its security vulnerabilities in a more familiar context to the users (The City Metaphor). [6]

The goal is to provide experimental evidence of the viability of this 3D modelling Then it is needed to consider how to implement 3D model visualization according to the given source code. [7]

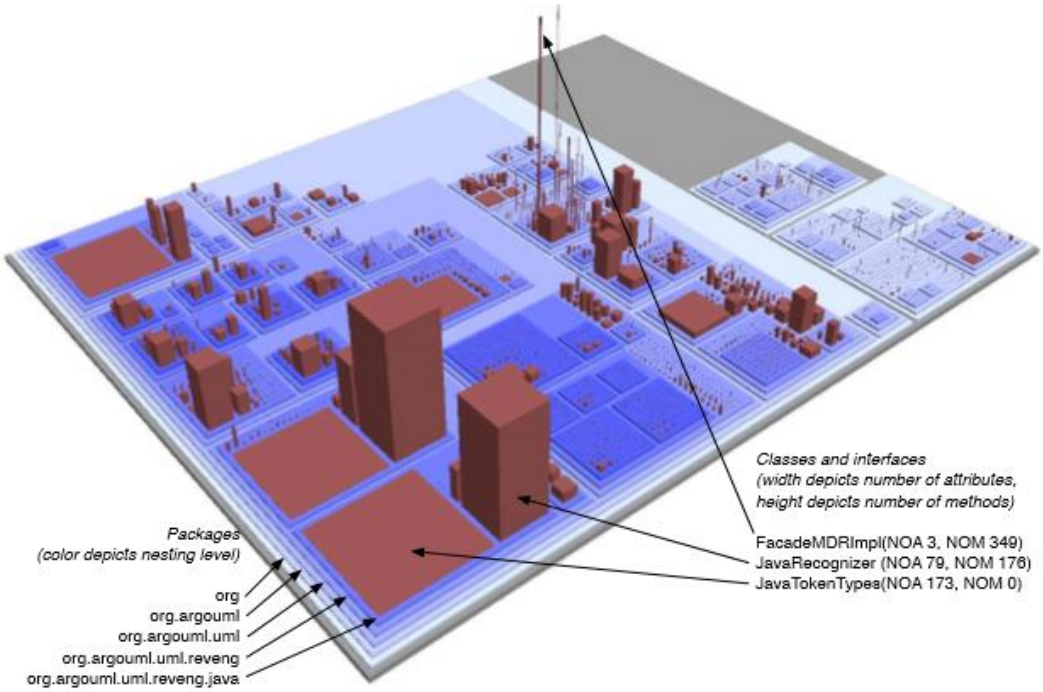


Figure 2.4 An overview of the city of ArgoUML v.0.24

2.4.4 Visualization of Code Evolution With Vulnerabilities

When considering the relationship between code review coverage and post-release defects, review coverage is negatively associated with the incidence of post-release defects. However, it was only provided with significant explanatory power to some of the studied releases, suggesting that review coverage alone does not guarantee a low incidence rate of post-release defects[13]

Source code is changed many times during the life cycle of a software system. This will lead to the problem in which developers may not be able to get an insight into these changes. Just by looking at the changeset it's difficult to get an overall idea about the changes that have been done throughout the project and there is a clear need to develop a tool to represent the software versioning via 3D model [8]

The Code Evolution and vulnerability changes visualization is very useful to show how the vulnerabilities changes and when new methods are created and disappear. Some evolutionary

patterns can be found, for example, a building (class) that evolves and loses an ever-increasing number of bricks (methods) looks unstable. Another example is when a large number of bricks are suddenly added from one version to another and new vulnerabilities arise because of that. Correlating the timeline visualizations of several classes and vulnerability changes enables the detection of causes for vulnerabilities and massive refactoring [33].

Hence it's clear that it is necessary to keep the track of the vulnerabilities identified over the project life cycle and only keeping those records is not enough to cater to the idea about security aspects of the project.

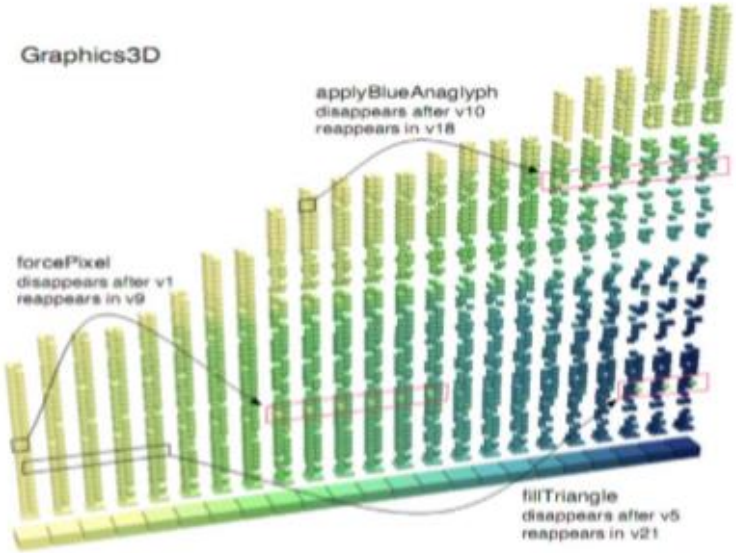


Figure 2.5Evolution of the "Graphics3D" class of the Jmol software

2.5 Conclusion

Software Visualization	Static Code Analysis Tools
Security Vulnerabilities and their evolution are not visualized	Vulnerabilities are not given with its impact on the system
Doesn't support second level drill down	If the project is large and complex its very difficult to refer to the source of an issue

Table 2.4 Limitations of Current approaches

By analysing the above evidences conclude that there is not enough research that has been done on combining Vulnerability Analysis with the Visualization Mechanisms with Code evolution representation.

There are various software visualization approaches that have been used over the last two decades that have led to a plethora of visualization techniques and these can be classifiable into several taxonomies. Each technique targets one or more of a software system and represents information according to its own visual language. Performing an analysis of several aspects of a software

system(eg: Design, Evolution, Complexity) would require conducting separate analysis for each targeted aspect using different visualization.

While trying to select a suitable representation for software, several researchers proposed different representation techniques using real-world metaphors. These techniques use easily understandable elements of the world to provide insights about software. For example, codecity(techniques are based on a City abstraction), Metaballs(3D modeling technique which can be used to represent complex organic shapes)
According to the facts included in this section depicts that SonarQube is a code quality measuring tool which has been widely used in the software security domain.The proposed solution from this dissertation has used SonarQube for identifying OWASP Top 10 security vulnerabilities.

Chapter 3: Research Methodology

3.1 Introduction

The methodology is the way of handling the research question and it is addressed with the knowledge gathered through referring to literature review. As mentioned earlier, the scope of this research had spread among many fields in modern computer science. As a result of these reasons, many experiments and techniques had been taken into consideration to cater the goals of the research. As mentioned in previous chapters, the goal of this research is an efficient vulnerability detection, visualization and evolution of software systems using code city metaphor. This chapter describes a comprehensive overview of the implementation procedures which had been undertaken throughout the project.

3.2 Problem Analysis

The primary aim of this Project is to identity possible security vulnerabilities of a software system in earlier stage(While developing the system) and represent those identified vulnerabilities and evolution of these vulnerabilities in an attractive manner by using 3D visualization.To achieve this goal, research was done by exploring relevant research papers, dissertations and tools.By carrying out a background study, system requirements and Architecture have been identified. The limitations that were identified in this approach, information gained by referring concept papers were incorporated while working with the design of the system architecture.

Due to the security vulnerabilities and System design flaws, leads to major security issues as mentioned in Chapter 1. Because of this, code reviews play an important role in the software development life cycle. Static code analysis tools[1,2,4] were explored in order to elect a tool to identify the code-level security issues as aforementioned in Chapter 2.Using the literature review conducted on Vulnerability analysis tools, SonarQube was selected as the Static code Analysis tool aforementioned in Chapter 2.OWASP Top 10 was selected to get counter measures to vulnerabilities as it links with SonarQube.

However there is no direct approach exists in available tools to find the association between identified vulnerabilities and the part of the source code relevant to the vulnerability as mentioned problem definition in Chapter l, the research component was based on discovering an approach to map the detected vulnerabilities to 3D metaphors and provide relevant countermeasures to user and compare the evolution of the identified vulnerabilities between versions.

3.3 Design Constraints and Assumptions

The evolution of vulnerabilities of software systems from the Secure Codecity with Evolution can be used as a separate software application or a component of a software application. For the vulnerability identification we have used SonarQube.According to the background study conducted in Chapter 2,the number of vulnerability types identified by SonarQube is maximum for the Java Web Application project compared to the Other supported languages. Hence, the solution only supports the Java Web Applications which is compatible with the supported version of the Java language from the SonarQube.

The intended users of the Secure Codecity with Evolution are software developers who should have a basic knowledge about the software security in order to use the framework.Subsequently,

the user should have familiar with the SonarQube source code analyzing process due to the framework uses SonarQube for software vulnerability identification. The framework identifies bugs and categorizes them into OWASP top 10.Other kind of categorization methodologies are not available.

User should configure the SonarQube before using the framework.For the Visualization Codecity metaphor has been used since it's easy to represent the software metrics to the user.Codecity metaphor has been modified in order to integrate security vulnerabilities information and give user a clear understanding about these vulnerabilities of a software project.

3.4 Hypothesis

The below hypotheses have been formulated to cover the scope of the proposed research and to measure the effectiveness of the proposed Framework. By gathering some information from previous research, hypotheses were formulated.

Null Hypotheses	Alternative Hypotheses
H1o The overall accuracy of the answers when considering all tasks is similar in experiment and control group	H1 The overall accuracy of the answers when considering the tasks is different in experiment and control group
H2o The Usability ratings for the systems are same in the experiment and control group	H2 The Usability ratings for the systems are different in the experiment and control group
H3o Time taken to complete all the tasks are similar in the experiment and control group	H3 Time taken to complete all the tasks are different in the experiment and control group

Table 3 1 Performance characteristics evaluation

3.5 Selecting Sample Datasets

Systems accept the sources which are used in the Java programming language.Sample data sets were chosen from open source OWASP Benchmarking projects.In order to be a fair dataset, it should have followed globally accepted the practices and procedures used while developing these systems.Selected projects have been Licensed under MIT.

After the analysis, following source had been selected as input source

- Security Shepherd
- WebGoat(insecure web application maintained by OWASP for evaluation purposes)

Another reason behind this selection is these projects have especially been designed to discover the vulnerabilities and we know what are the vulnerabilities exists in these projects.Webgoat consist of known vulnerabilities that we can use to validate the Framework.

3.6 Design Overview

Ultimate goal of the project was to create a tool which is free and capable of analysing security vulnerabilities of the source codes and visualize the code evolution using vulnerability evaluation metices. Projects which were selected(WebGoat,Security Sheperd) for testing and analysis of the system are Apache Source codes,Reasons behind the selection are these projects are well-known

and follow general standards,Projects contains known vulnerabilities that can used to verify the correctness of the system.For visualization purposes, the Code city metaphor was selected. Proposed system has main components as follows.

Static Code Analysis
Vulnerability Processor - read, categorize and store relevant into the database.This contains several sub components(File Hierarchy Processor,Metrics Preprocessor,Issue and vulnerability processor,Color generator)
Evolution processor - Compare the vulnerabilities between different versions of the codebase and generate informations required
Visualization Engine(Building and District Generator) - which can be used by the developers to visualize the vulnerabilities with the association with the codebase using codecity metaphor

Table 3.2Components of the System

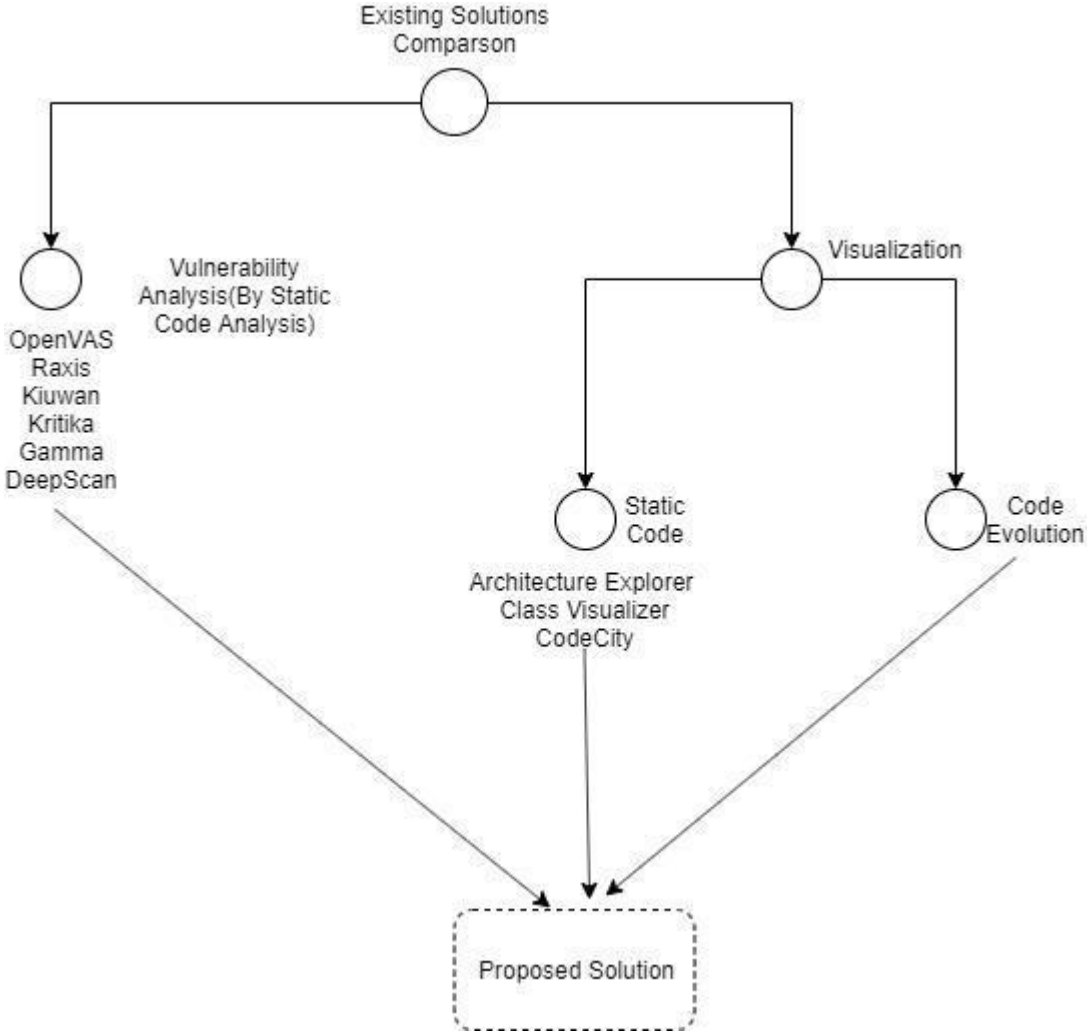


Figure 3.1Taxonomy of Comparing Proposed solution with Existing solutions

3.7 System Overview

As mentioned above, the system was divided into independent components. More focus and the weight were given for designing Vulnerability Processor,the Evolution processor and visualization engine, since those components provided high value for the end users. Building the vulnerability knowledge base of the analysed source codes was the major part of the proposed system stem and that was not completely automated. Some manual work also taken into consideration to continue the workflow of the building vulnerability knowledge base such as after gathering the source code samples. Those things were required to be uploaded to the static analysis tool to perform the analysis. Also when the analysis was completed by the tool, a false positive removal was performed to make the result set accurate. After that the result was imported to the system in a particular format which it could be interpreted by using Vulnerability and it was stored in the database for using for evaluation. Formed Visualization model will be transferred into the Visualization Engine(FrontEnd).

- Upload the selected source code sample to the static analysis tool
- False positive analysis
- Generate 3D visualization (Vulnerability Evolution) based on analysed versions of a project

Below is the high-level overview of the complete System. The diagram shows all the components of the proposed system and how each component is going to interact with other components to provide the necessary output of the proposed system.

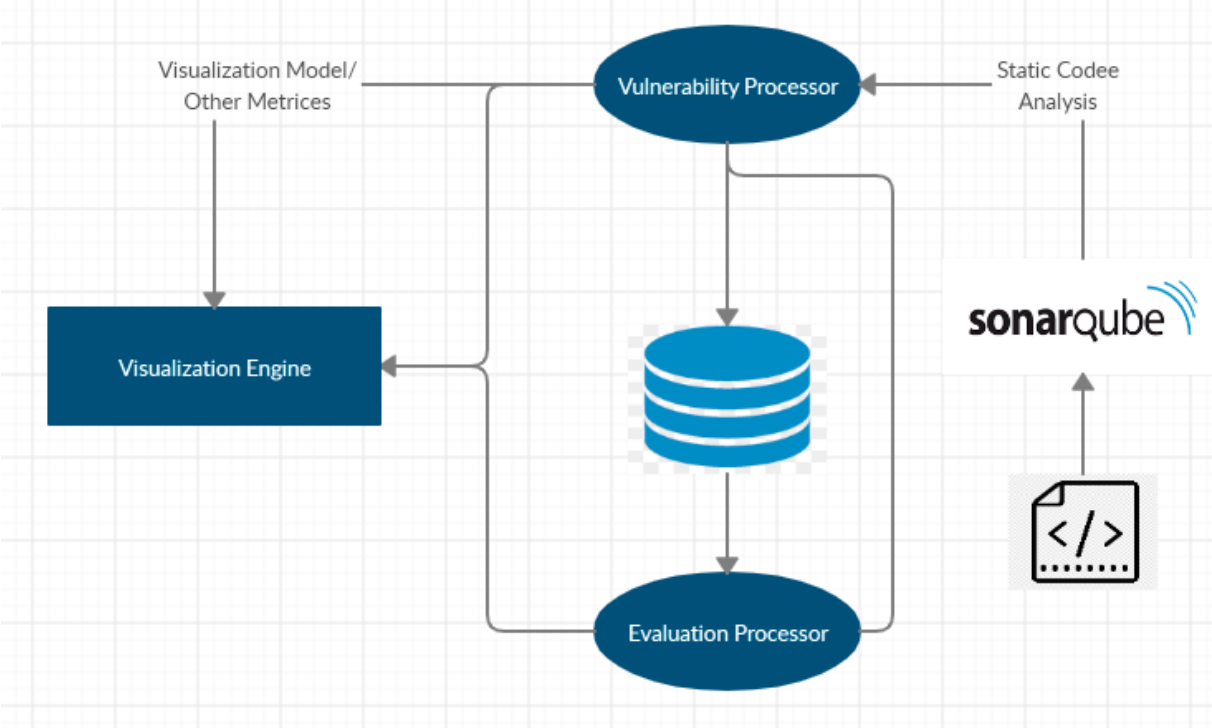


Figure 3. 2 System Overview

3.7.1 Vulnerability Processor

This is one of the important components of the project and it could be used by developers to analyze the potential vulnerabilities of source code. Vulnerabilities were analysed and the ways to address those vulnerabilities were found, prepare the visualization model.

A commercial static analysis tool was required to perform vulnerability assessments of the source code samples. Also it was practically impossible to purchase a commercial tool for the project due to the pricing of these tools(These tools are very expensive).For an example static analysis tool like kiuwan cost between $600 to $2,550 based on the project.During the static analysis selection process, mainly the analysis was done as a part of the project to understand the features , capabilities and the differences of the static analysis tools and ease of use. Most of the Open Source tools accuracy level was not satisfactory.SonarQube has been selected as the StaticCode Analyzer according to the conducted analysis Chapter 2.

After analysing the required vulnerabilities using SonarQube,Other required details for the visualization(Details required to generate basic structure of the city) ,class level details,Metrics processing,File hierarchy processing will be conducted.

The codes were analyzed by the code processing tool(SonarQube) and they have categorized accordingly. The tool can be connected to the created knowledge base to analyse the potential security vulnerabilities of the selected source code sample and the feedback was given to the developer in a user-friendly manner. Color code generators decide the colors for the buildings based on the Security Vulnerabilities Severity, Cognitive Complexity, and the Remediation effort. Identifying the associations between codebase and vulnerabilities will be done by combining the knowledge gathered on file hierarchy processing and vulnerability processing.By combining the remodation suggestions generated by SonarQube and User Feedbacks tool is capable of generate more detailed suggestions to resolved the vulnerabilities.

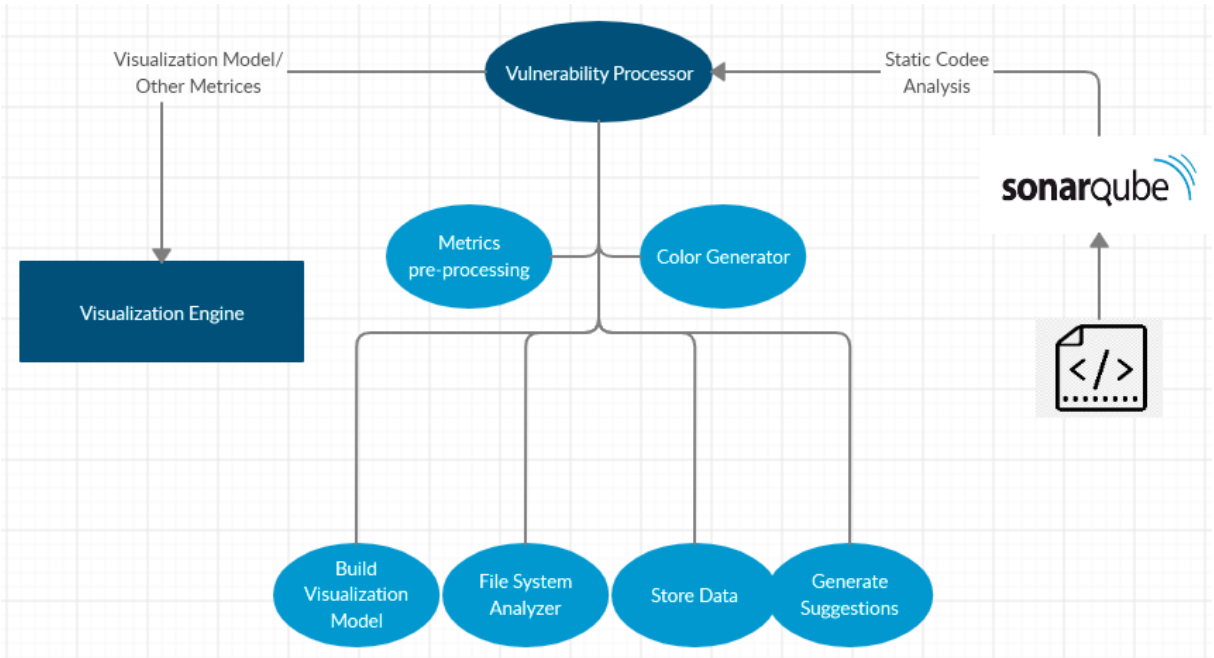


Figure 3.3 Main Tasks of Vulnerability Processor

Information gathered during the analysis will be saved on the database with reference to its version(Project version), which will be used in analyzing the vulnerability evolution over the different versions of the codebase.

3.7.2 Visualization Engine (Building and District Generator)

These processed data (Abstracted details from Analysed code base) were fetched into the visualization component. Here the abstracted details will be modeled into the Visualization model which mapped into a 3D city using code city metaphor.

3.7.3 Evolution processor

Process vulnerabilities between two different versions of the codebase and Generate the Vulnerability Evolution Model. Combining the information's changes between two versions generated by SonarQube and combining it with Pre Analysed data Evolution Model will be generated

Following describe the Secure CodeCity with Evolution Framework Approach and detailed description about components

3.8 Secure CodeCity with Evolution Framework Approach

The task of discovering security vulnerabilities inside the source code can be done using SonarQube without having much difficulty.Even it could identify the files which contain them, having a comparison to get an overall idea of these vulnerabilities is little bit challenging. In the current approach we have to get vulnerabilities of each and every file and have to calculate the vulnerabilities separately. It is very time consuming. Although we can find the vulnerable classes or methods, we have no idea about which class or method should we have to give the priority.It's even worse if we are trying to do a comparison between different versions of the codebase. To address these issues, we introduce Secure CodeCity with Evolution, a new 3D code visualization tool that aims to improve a programmer's understanding on security vulnerabilities of an existing codebase in a manner to get an overall idea about vulnerabilities, countermeasures.

The objective of this study is to improve the understandability of system owners in the perspective of software vulnerability via analysing the code evolution through various versions of the project. The solution was carried out in three modules. In the first module, software vulnerabilities were identified through static code analysing techniques ,abstracted details will be visualized using codecity metaphor. In the second module is a drill down view where the users can use to discover further details, Finally in the third phase, the software code evolution was visualized between different versions of the code base.

The proposed solution is an extension of Detection and Analysis of Software Security Vulnerabilities [2].Secure CodeCity with Evolution organizes source code into a 3D scene in order to take advantage of human spatial memory capabilities and help one better understand. By extending 3D space into more levels, Secure CodeCity is also able to provide an exciting game-like environment, thereby encouraging engagement and subverting boredom. Secure CodeCity also supports two unique points of view: exocentric and egocentric, which allows one to examine the vulnerabilities at different granularities. In method level, different charts are used to present different granularities of vulnerabilities inside a class.

We aimed to create a vulnerability visualization tool that helps users in becoming familiar with vulnerabilities in existing codebase and comparing it with different versions. This tool must be easy to work with and must show information in a form which reduces the user's cognitive load. One thing that sets Secure CodeCity apart from current tools in the literature is that it is designed to be suitable for both beginner and experienced developers alike.

Software visualization enables the user to interact with a representation of something familiar, namely a world with familiar objects that a person can interact with may help to better explore software structure[34].The visual environment as a more suitable option for teaching beginners. Also, Secure CodeCity with Evolution offers code interaction from an exocentric and an egocentric perspective, combining the benefits of both interaction modalities.

3.9 Secure Code city with Evolution Architecture

As mentioned above, the framework consists of three related views,which are First level ,Second Level and Evolution View.First Level is the initial view where the abstract details of the system being visualized. Second level view visualize deeper but more restrictive scoped vulnerability information related to the input. In the Evolution view where the user can view the evolution of the vulnerabilities between different versions.

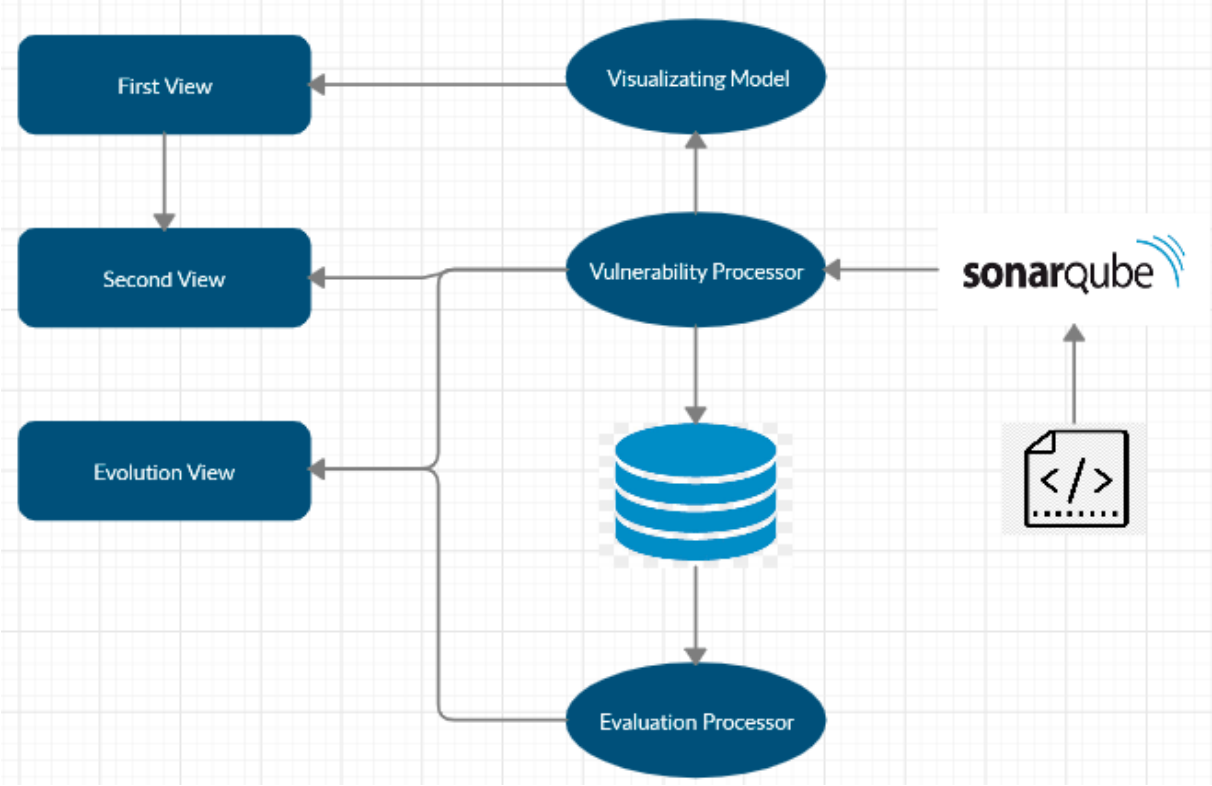


Figure 3.4High Level View of Secure Code City with Evolution

3.9.1 First View of Secure CodeCity With Evolution

This is the initial view from where the user can get particular security vulnerability information related to the source code.In the First level view user can will be able to visualize the source code of the input project in 3D city.where each building depicts a class of the input project.The footprint size of a building is determined according to Cyclomatic Complexity of the corresponding input source file.Where the height of a building represented according to number of lines of the source code file.The color of the building which represents a class varies according to the overall severity level of identified vulnerabilities and Security Rating.Further Total number of Vulnerabilities ,Total number of issues will be available in the First level visualization

3.9.2 Second View of Secure CodeCity With Evolution

When a user selects a building in the first view ,a second view appears .This view also has buildings which represent the methods in the class related to the selected building.The color of the building varies according to the overall severity level of vulnerabilities in the particular vulnerability. This view can identify what OWASP vulnerability categories are included in the selected class.In here users can identify the number of Major,Minor,Critical vulnerabilities that exist in the selected class.

3.9.3 Evolution view

In this view users can view the evolution of the vulnerabilities between two different versions of the codebase.Using this view user can identify addition of new components,and changes in the existing components to the system affecting the system security aspects.

3.10 Sub-components of the Framework

l . Metrics Pre-processor

Ths modules accumulate the information of different metrics (number of classes,number of methods , names of methods, starting line and ending line of a particular method,lines of codes) of classes and methods.Processed information by Metrics generator are used as an input to the building generator and vulnerability processor.

2. Vulnerability processor

This module is used to get vulnerability details of the classes of a project. It provides details related to the particular vulnerability like severity, message, debt, effort and so on. These values are extracted from SonarQube.

3. Building Generator

This module is used to generate models which have been used to 3D building views for the methods of the selected class. Each building represents a method. The height of the building shows the number of lines of the method.The footprint size of a building is determined according to Cyclomatic Complexity of the corresponding input source file. The metrics for generating buildings were taken from metrics Pre-processor.

4. Sonar Vulnerability Summarizer

This module is used to summarize different kinds of information about the projects.No of vulnerabilities,Vulnerability types,the severity level of vulnerabilities,etc will be generated by combining the metric pre-processor information and other project related matrices. Different types of charts will be generated after processing the information

5. File System analyser

This module is used to analyse the files structure and find the associations between vulnerabilities and Classes,methods.This component merges the information gathered by other components which is used to generate the Visualization model.

6. Color Generator

By considering the level of risk factor,the Color generator module was utilized to provide colours for the building.For determining the severity level of vulnerabilities in each method, OWASP Risk Rating Methodology was taken. numerical values were assigned for parameters (Ex: Exploitability, Prevalence.) that were considered for rating the issue category. The method which did not have the vulnerability, was colored in green. Method which has vulnerabilities but not providing any owasp category related to this, it was colored as brown. The other methodologies which possess and Dotnet core and tested using Jasmine and Karma . Frontend of the project is built using Typescript,Javascript programming language with HTML and CSS

Chapter 4: Implementation

4.1 Implementation Overview

After design of the project was completed, the next challenge was to implement the project and also make sure implementation would achieve all the project requirements specially user-friendliness and accuracy. Most importantly implementation should not limit or restrict the required features of the project and implementation enchase or facilitate to enrich the project features. Certain decisions should be made to achieve the successful implementation of the project, including underlying the technology frameworks that needed to be used and back-end technology which is going to be used. The primary focus of this research was to implement the project successfully rather than using the best and the latest technologies available in the industry.

4.2 Tools and Technologies

The framework was developed as a stand alone application which render in the browser. It consists of a different levels and some levels are used3D visualization .Threejs library is used with JavaScript to 3D implementations.TheFirst Layer of the application is built using TypeScript, Angular and Dotnet core and tested using Jasmine and Karma . Theres of the project is built using Typescript,Javascript programming language with HTML and CSS

The first layer of the application has developed using TypeScript,Angular,and Dotnetcore because the Secure CodeCity with evolution should run fast and reliably to visualize the security related data. TypeScript is the superset of JavaScript which includes some of the features includes in oop based programming languages. The main advantage is its supporting for spotting common errors on real time. First layer of the application was divided into components such as vulnerability summary, city builder, scene. Sidebar, top bar etc. and implemented those components using Angular and RxJs.. Angular Testing Frameworks (Jasmine,Karma) are used to test the application frontend and for backend Nunit,XUnit is used. Also CSS and Sass were used to style the front end. Webpack used to bundle Javascript files for usage in a browser.

The metrics pre-processor for method level was built using Dotnet core programming language with Dotnet core webapi related technologies. The main reason behind choosing dotnet core is ease of implementation and rich set of libraries available. MsSql is used for creating the database and other required data transferring operations. The additional reasons , and for the main development communicate with 3D techniques and allow to letting JavaScript programming language is, it is in browser a popular, robust, secure run implementation of the used for the chapter explains the development approaches taken in the implemented framework described in Chapter 3 with the tools and technologies .A detailed description of the implementation of each common CodeCity Framework architecture is described under sub sections

4.2.1 Static Code Analysis: SonarQube

In order to identify the Security vulnerabilities of a particular software application and of software metrics, the Secure CodeCity Framework user needs to analyse the source code using SonarQube.

The vulnerabilities identified as security bugs categorized with respect to OWASP 10[13] by this tool. SonarQube is a static code. It consists of different levels and some levels are used for 3D visualization . Categorization of software bugs into OWASP T10 is an additional reason for selecting this tool for the proposed approach of the framework as identified in Chapter 2.

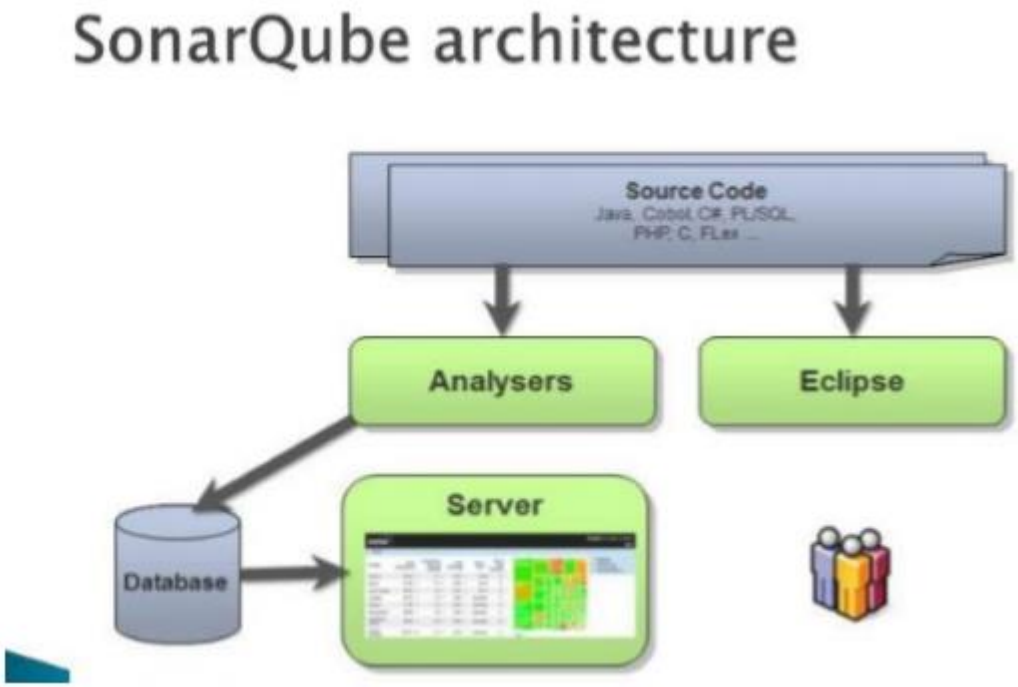


Figure 4.1 SonarQube architecture

4.2.2 Translation Middleware

This component acts as a bridge between a SonarQube Api, database and Frontend Application. This Middleware has the ability to derive the abstract syntax tree(AST) related to a Java source code as well as, to breakdown Java source code to methods, statements with particular keywords etc and process them to find associations between different metrics.

4.2.3 First Level (Class Level)

The First Level (Class Level) of the application is built to map project to a 3 dimensional City metaphor Security Vulnerability Severity, Security Remediation Effort Security Vulnerability Rating Issue, cognitive and Number of developer details could be taken using the colour spectrum of the building Also specifies source file details such cyclomatic complexity, number of lines of code Security Remediation Effort ,Security Vulnerability Severity , Security Vulnerability Rating ,Cognitive complexity and number of developer could be taken by selecting the building. Application is built as a Single Page Application using Angular and ThreeJs.Functionality of this component has been Tested using Jasmine and karma

1. **Metrics Pre-Processor**

Using SonarQube provides API we can obtain the different types of metric details related to the scanned projects by sonar Scanner. Metrics pre-processor process these metrics and bind them to the Processing Model. This model is created to represent the to store details about the extracted details from SonarQube. So in order to make it efficient and speed up the process, APIs are being called in Asynchronous manner. Measures such as cyclomatic complexity, number of lines of code

are needed to map files to buildings and number of file details are needed to map components (packages) to districts. These generated details are fetched to the Model builder to build secure code city models.

For implementing the required functionalities in the the api dotnet core has been used ,For the testing api Postman is used

2. **Vulnerability Processor**

SonarQube provides a facility to extract vulnerability details of any given project via SonarAPI. It sends output details as a JSON object. Then details are processed according to various needs of the project.

For implementing the required functionalities in the the api dotnet core has been used ,For the testing api Postman is used

3. **Building and District Generator**

The metrics results related to class details and package details would be stored in the "Visualization model" which uses the "Processing Model" in the metric pre-process stage. All building, Packages related metric details would be taken from the Processing Model.

"Generate CodeCity" Component rules have been defined i.e. mapping of line of code in source file to height of the building, mapping of cyclomatic complexity of source file to footprint of the building etc.

The Visualization model will be used to generate the city in the "Generate CodeCity" Component. The Functionalities of those components were implemented using TypeScript and for generating the codecity view ThreeJs has been used.Unit tests are implemented using Karma and Jasmine

4. **File Hierarchy Processor**

Build the association between files and identified vulnerabilities file Hierarchy Processor is used**.** Associations between districts, buildings related packages, files, identified vulnerabilities ,Classes, methods are identified and use these information's to update the "Visualization model"

For implementing the required functionalities in the the api dotnet core has been used ,For the testing api Postman is used

5. **Colour Generate**

Colour code generator will decide the colour code for a particular class by processing the vulnerabilities and other metrics(Exploitability, Prevalence).These generated color code will be used to update the Visualization Model

For implementing the required functionalities in the the api dotnet core has been used ,For the testing api Postman is use

4.2.4 Second Level (Method Level)

1. **Metrics Pre-Processor**

SonarQube API does not provide a way to get method details inside a class. In order to get the methods details, source code will be fed to java parser and extract method name, size and method lines details.API is implemented using dotnet core to send those pre processed metrics values as JSON format to building the generator component

2. **Building Generator**

The metrics results related to method details which are coming as a JSON object from Building generator api will be processed to extract the required information to build the view using Typescript. Some vulnerability details are also needed to build this view and those details are extracted from vulnerability processor. View is generated usingThree.js library.

3. **Colour Generate**

Colour code generator will decide the colour code for a particular Method.

4.3.5 Evolution View

Users can use this view to get information about how the changes between different versions affect the software security. User can select two different version of a software which was analysed by the framework. SonarQube API provide information's regarding version changes. Evolution Processor will combine SonarApi metrices with previously analysed result which was stored in the database. Processed data will be bind to the "Evolution Model". For implementing the required functionalities dotnet core has been used and NUnit has been used to prepare the unit test cases.

"Evolution Model" will be transferred to the "Evolution viewer" component in the Angular Project. Typescript has been used to implement the functionalities, and for 3D visualization of the Evolution Threejs has been used

4.4 Summary

The chapter above, presented the procedure of implementation of the framework(secure code city with evolution) with the tools and technologies in each component and Implementation of the design modules in secure code city with evolution Framework architecture explained with technically presented as an integration of them with the reasons behind in selecting relevant tools and technologies.

Chapter 5: Testing and Evaluation

5.1 Introduction

This chapter will describe the results of the project, application features, functionalities and capabilities to evaluate the project and also possible approaches to test and verify the application functionality to Make sure it provides the expected quality output. Below step was used to evaluate the developed application.

- Source code testing to make sure there are no errors and logically it is implemented as expected.
  - Angular Unit Tests (Using Jasmine and Karma)
  - Web Api Unit tests(Using NUnit and XUnit)
- Functionality testing.
  - Functionality testing of the vulnerability explorer(Static code Analysis)
  - Functionality testing of the Vulnerability Processor
  - Functionality testing of the 3D Visualization
  - Functionality testing of the Visualization Engine(Building and District Generator).

- Usability testing.
  - Verify whether users are able to understand the framework functionalities without extensive training
  - Verify whether users can increase their awareness about the system vulnerability aspects by using the framework

5.2 Testing

The testing procedure was conducted as a strategy to ensure secure CodeCity with evolution product operates as intended in the specification. It can be realized under two major categories namely, functional and non-functional testing .In Functional testing, it includes unit testing , integration testing and system testing to verify that the implemented framework functions correctly and provides the results in accordance with the development constraints. Unit testing for the front end was performed using Jasmine and Karma. The First Layer View of the application was built using Angular,Typescript,JavaScript and Threejs, which was tested using Jasmine and Karma and the backend developed using dotnetCore and tested using xUnit and NUnit. API calls were being tested using Postman. The manual testing also carried out for the entire system to check the functionality of the whole system by writing the test cases.

Acceptance Testing was conducted under functional testing where the System was Evaluated with the help of industry expertise and their experience gathered to verify that the System works properly and meets the requirements. System Testing was performed by analysing benchmark projects with SonarQube itself and the whole system and check weather system generates the expected outputs to verify that all components together risks properly

Performance Testing was conducted under non-functional testing and the framework was tested for analysis of large-scale projects to check whether the system crashes or fails to produce expected outputs. The OWASP Benchmarking projects namely Security Shepherd and WebGoat(insecure web application maintained by OWASP for evaluation purposes) used for the evaluation purpose.

5.3 Evaluation

This research intended to demonstrate a novel method and a proof of concept of framework proposed for visualizing security vulnerability information and its evolution throughout the SDLC of a java projects. First evaluation is done to measure the overall accuracy of secure code city with evolution. Second it was evaluated whether the results generated by secure code city with evolution is time efficient.

5.3.1 Validate Functional Requirements

For evaluating the functional requirement Security Shepherd and WebGoat has been used as benchmark projects

- The proposed tool was executed against different versions (Ex:WebGoat 7.1,8.0) of the Source Codes(Dataset) and vulnerabilities were identified.
  - Testing data set has known vulnerabilities

- Each version was tested on security vulnerabilities that had been used for evaluating
  - Ex: Which areas of code will make the system to be exploited by the SQL Injection techniques

- Test cases were prepared to evaluate the productivity and correctness of the system for each and every version.

- Using these benchmark projects, the Correctness of the predicted vulnerabilities were verified.

- 3D code visualization was checked for each version with identified vulnerabilities
  - Here, 3D model was verified to see whether the generation works properly
  - Did it work as expected?
  - Were we able to go through different versions of the system and identify vulnerabilities?
- System was checked whether it had all its functional requirements
  - Did the System have all the functionalities that we were trying to achieve ?

Ex: If there was a vulnerability in some class, its representative building should be in a different colour. User should have the able to click on it and identify what was the issue

- System suggested solutions were checked to resolve some vulnerability validity.

5.3.2 User Evaluation Experiments

The User Evaluation experiment was designed based on the Subjects Design, which is one of the famous experimental design methods in software Engineering. The main purpose of the user evaluation experiment was to use a multi-fold evaluation of secure CodeCity in terms of accuracy when conducting tasks, time to complete tasks ,correctness of the tasks and overall usability of Secure CodeCity when compared with SonarQube.The research population for this experiment had 20 subjects who had 2-10 years of working experience in the industry.Subjects has been

selected based upon on their industry experience and familiarity based upon the source code reviewing tools like Sonar Qube.

All the subjects have been divided into two groups one as Experimental group and one as the control group.While grouping the members, we have considered the factors of industry experience and intimacy of the source code reviewing tools.Figure 5.1 and 5.2 shows the distribution of these factors among two groups.

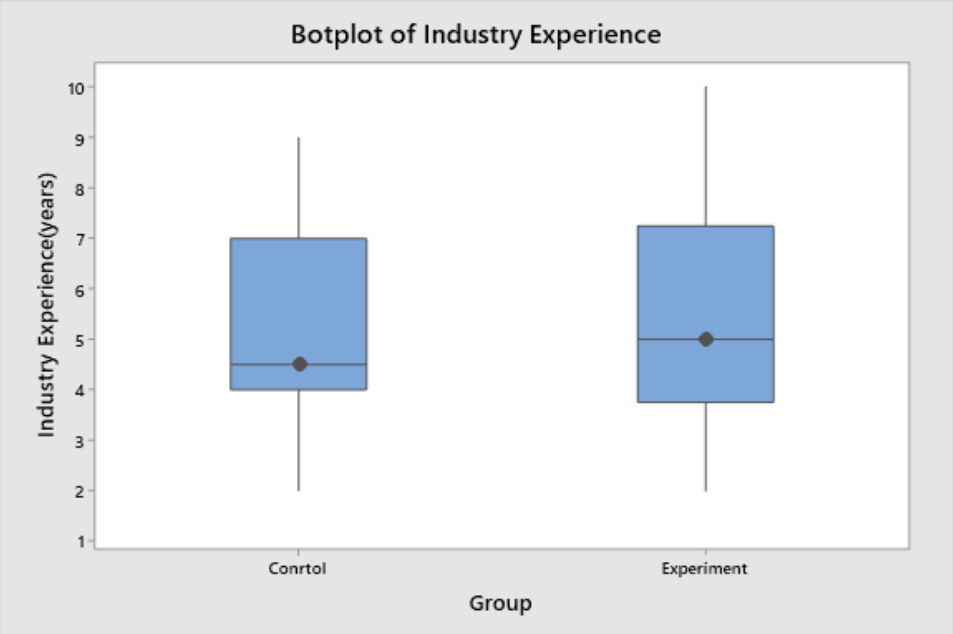


Figure 5.1 Boxplot of Industry Experience between Experimental and Control Group

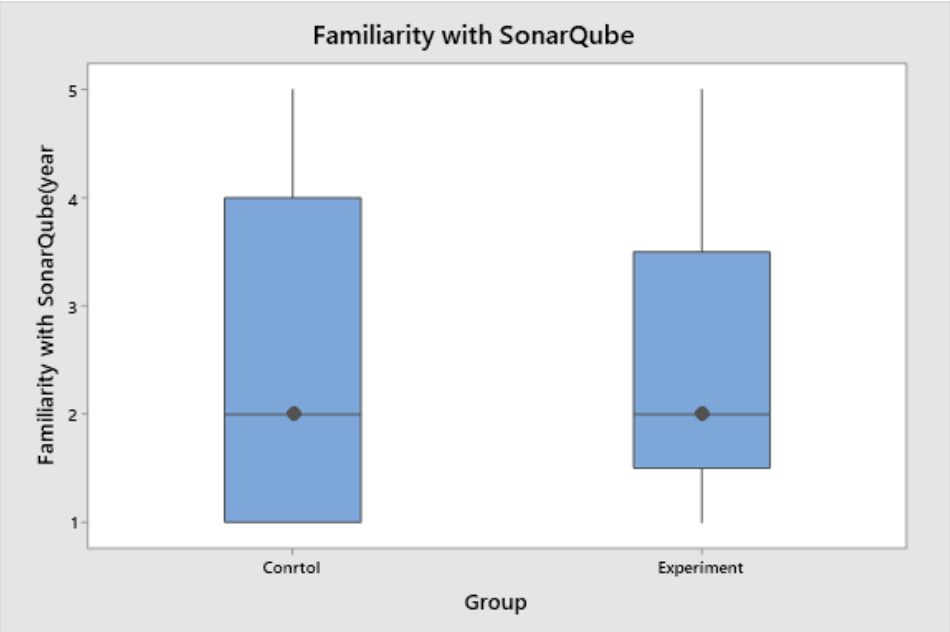


Figure 5.2 Boxplot of Industry Experience between Experimental and Control Group

The Experimental group has been provided with Secure code city with evolution while control group has been provided SonarQube 8.3

According to the Hypothesis defined in Chapter 3, questionnaires has been formed.The experiment has been conducted after an explanation about the secure codecity with evolution and communicating the objectives of conducting the experiment.Afterwards simple demonstrations

have been conducted inorder to give the users proper understanding about the framework and the Sonarqube.Then the questionnaires has distributed to the participants,where the participants asked to answer upon completing each and every task within a defined time frame(10min and 12min).

The experiment group has been instructed to answer the questionnaires based on the experience they had while working with the Secure code city with evolution while working with the task list. The control group has been instructed to answer the same questionnaires while working with the SonarQube on the same task list. After completing the experiment, overall usability has been measured from the each group.

5.3.3 Selection of Questions and Scenarios

While selecting the scenarios and questions we have focussed on covering up several critical aspects of software engineering and evaluating. For the tasks, two open-source apache projects have been selected. Questions have been prepared to evaluate the points mentioned in the hypothesis and evaluate whether the objectives have been fulfilled.

We have formed questions which fall into three categories identified in the context of software visualization, evolution and vulnerability identifying. These categories will cover up the three levels in the framework. First questions are simple and straightforward where the subjects can answer without having that much analysis. It included questions like

- Which class contains the most number of vulnerabilities?
- What are the number of vulnerabilities contained in a particular class ?
- Which will provide information which will be useful for the software engineers, system architects, business analysts , project managers to evaluate the software.

Second Question set focused on explore the method level vulnerabilities and details of various security aspects.Next set of questions are prepared to evaluate the understanding related to software versioning and vulnerability changes according to that. Example questions are like

- What class has been more vulnerable since the new features?
- How the new changes are going to impact the secureness of the software ?

While we are selecting the questions, we were careful to avoid biases towards the one system.

5.3.4 Selecting Sample Projects for the experiment

Two open-source Apache projects have been selected for conducting the experiment. These two selected Apache projects(Security Shepherd, Web Goat) which are available on the GitHub have been used by the researchers over past years. Projects have been created to enhance awareness of security among a variety of skill-set demographic. The second project which was selected is Web Goat. Web Goat is a deliberately insecure web based application, which is governed by OWASP, which was designed to teach security lessons of web applications. The project's selection was mainly based on the availability in the public instances of SonarQube and Jenkins, which have the ability to do all the tasks. Also we ensure that the selected projects are of a reasonable size due to the time limitation in conducting experiments

5.3.5 Selecting Experimental Subjects

The rationale behind the selection of experimental subjects is that the experiment subjects should have the basic knowledge regarding static code analysers which are required to perform the given tasks. While conducting the experiment subjects have been provided with simple introduction about Secure code city with Evolution , SonarQube and the selected projects

5.3.6 Formulating Questions

While formulating questions we have considered three aspects. Questions have been developed based on those aspects.

Category 1 : To explore the class level vulnerabilities and related details

By conducting a vulnerability identification process, will help programmers and non-programmers to get the overall picture of the severity levels of each class, and ultimately as a united whole project by aggregating the security vulnerabilities with other project metrics. Showing the overall picture of the severity levels of each class is even more useful, so the most vulnerable areas of the project can be identified with ease. Existing static code analysis tools which available only keep track of the vulnerabilities of each class and the related details to those vulnerabilities, in a list view. If it can generate the security vulnerabilities of a particular class, it can be useful for a programmer.

Following 5 questions has been prepared to provide the insights related to software vulnerability for software practitioners, in class level.

Q1. Which one contains the most number of critical vulnerabilities of the input project, according to the system?
Q2. Which one is/are the most vulnerable class (*) of the input project, according to the system?
Q3. How many vulnerabilities are there in class UncheckedEmail.java?
Q4. What is the line of code(loc) value of class UncheckedEmail.java?
Q5. What vulnerabilities can be identified in class UncheckedEmail.java?

SonarQube can be used to observe the continuous code quality of a software system. However, in SonarQube, the vulnerability details are mutually exclusive from each other. There are no links between vulnerabilities or there are no aggregations created using those vulnerabilities, other than the fact that those vulnerabilities are categorized under corresponding class. For example, to find the answer to Q3, a particular user needs to up the SonarQube service, particular project should be scanned into SonarQube local server, the vulnerabilities related to the particular class (In this case, UncheckedEmail.java) should be identified, the remediation efforts of all the identified security vulnerabilities in the class should be added together in order to come up with the remediation effort of the class.

In answering the questions in Scenario 1, Secure CodeCity also fetches data from SonarQube. For this reason, it can be argued that the experimental results obtained from both the experimental group and control group are not significantly different in terms of correctness and the time. However Secure CodeCity performs more tasks behind the scenes in which users would take a lot of time in finding relevant vulnerabilities and analyzing those vulnerabilities to derive the ultimate answer. Because of this, it can be observed and concluded that experimental results obtained from both the experimental group and control group are significantly different in terms of correctness and the time.

Category 2: To explore the method level vulnerabilities and details of various security aspects

As showing the overall picture of the severity levels of each class, showing the severity level of each method which resides in a selected class is also important to programmers and non-programmers. The current practice is that the static code analysis tools only keep track of the vulnerabilities of each class and the related details to those vulnerabilities are represented in a list view. Vulnerabilities of each method are not shown separately. SonarQube does not provide a

direct way to get various security related aspects like getting OWASP related issues, getting MINOR, MAJOR, BLOCKER, INFO and CRITICAL issue percentages etc. There can be moments where the details of security vulnerabilities of a particular method can be useful for a programmer. To effectively analyze this scenario, three key questions have been identified (Q6 to Q8).

Q6. What is the most critical method in class Register.java?
Q7. What is the percentage of MINOR issues in class Register.java?
Q8 How many security vulnerabilities are there in method in classRegister.java? .

Experiments were conducted on both of the aforementioned projects. Answering Q6 and Q7 without Secure CodeCity with Evolution is cumbersome, requiring accessing each method in class and getting a number of issues with the related (OWASP tag. Then ranking value should be calculated for each and every vulnerable method and have to find the most critical method as the answer for Q6. Q7 also has to do some calculations to get percentage value after adding each and every type of same type issues together. Q8 can be done by getting the summation of vulnerabilities of relevant classes.

In this scenario, the advantage of using Secure CodeCity is that the ability to navigate to the exact line of a particular code in the IDE, which is also the starting line of a particular vulnerability without worrying about the locating procedure.Hence,different types of vulnerabilities can be resolved in less time.

Category 3: To explore information related to vulnerability evolution

Typical static code analysis tools have the capability of providing information related to changes in different version.Q09 to Q12 questions have been prepared to evaluate the capabilities of the systems while analyzing the vulnerability changes between particular versions

Q09.How many vulnerabilities are resolved between version 6.0 and 8.0?
Q10 .What are the classes that the vulnerability density has been increased ?
Q11.What are the new vulnerabilities occured version 8.0 compared to version 8.0 in class UncheckedEmail.java?
Q12. Calculate the vulnerability evolving ratio by comparing two versions ?

These questions have been designed to evaluate the ability to analyze the evolution of security vulnerabilities in different versions.

5.4 Data Collection and Analysis

In this section, we overview of the data collected to evaluate three hypotheses and the results taken through statistical analysis.

5.4.1 Overall completion time of the questions

Before starting the experiment we have advised the subjects not to spend more than 10 minutes for Q1-Q8 and not to spend more than 12 minutes for Q9-Q12. At the end of each task, they were requested to write down the time spent on each task. Average completion times for all the questions across the experimental and control groups presents in Table number 4. It was observed that overall completion time of the experimental group is less than that of the control

group. A notable advantage of Secure CodeCity was not witnessed in answering Q4 and Q5, in particular. However, Secure CodeCity significantly surpasses the SonarQube in answering Q1 , Q2,Q3,Q6, Q9,Q10,Q11 and Q12.

Question	Experimental Group	Control Group
Q1	11.52	438.46
Q2	12.14	435.64
Q3	30.71	170.65
Q4	78.75	68.55
Q5	34.28	25.85
Q6	11.68	349.25
Q7	16.25	34.82
Q8	17.50	55.25
Q9	58.75	478.50
Q10	38.25	585.75
Q11	25.25	245.75
Q12	85.25	578.25

Table 5.1Completion Time for The Questions in seconds

The box plots in Figure 5.3 shows the distribution of time across the experimental group and the control group, denoting that Secure Code City with Evolution was capable of obtaining the result much faster than that of baseline tools.

Secure code City with evolution's mean overall completion time is 297.73s , and it is 907.09s lower than the control group mean score of 2204.82s.These values clearly indicates the advantage box of secure Code City with Evolution over SonarQube, regarding the efficiency.

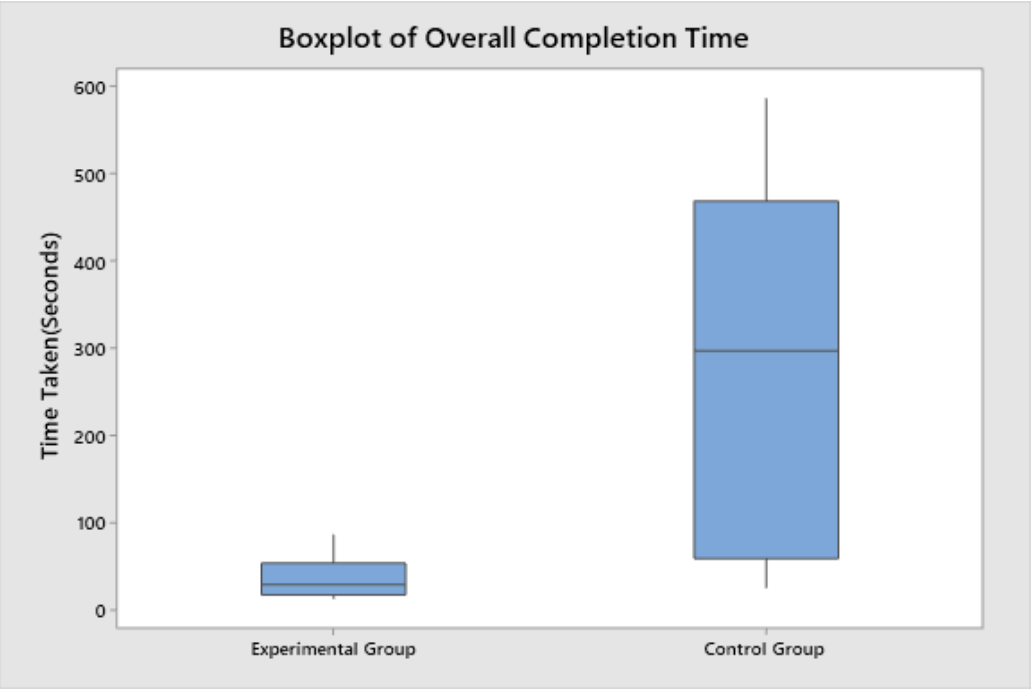


Figure 5.3Boxplot of overall Completion Time between Experimental and Control Group

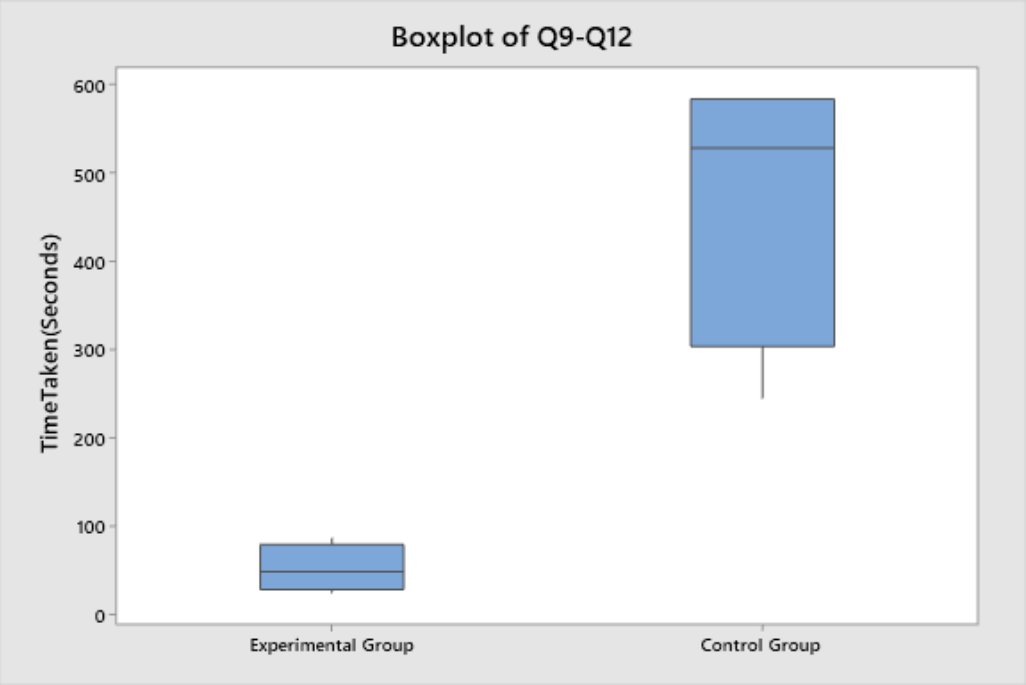


Figure 5.4 Boxplot of overall Completion Time between Experimental and Control Group considering only Q9-Q12

5.4.2 Overall usability score of the tasks

In order to evaluate the usability of Secure CodeCity compared with the SonarQube in performing the selected 12 tasks we have used 3 Metrics recommended by ISO/IEC 9126-4 Metrics.Which are Effectiveness,Efficiency,User Satisfaction.

5.4.2.1 Measuring Effectiveness

In the process of measuring Effectiveness we have considered the Successful Completion rate of the task using the following equation.

Effectiveness = Number of tasks completed successfully x 100%
Total number of tasks undertaken

Effectiveness of Secure CodeCity with Evolution(63.33%) is greater than SonarQube's 55%

5.4.2.2 Measuring Efficiency

In Order to Complete The Assigned Tasks Secure code city with Evolution has taken average time of 438.2 seconds while SonarQube has taken average time of 3466.2 seconds.Its clear that in efficiency Secure code city outperforms the SonarQube.

5.4.2.3 User Satisfaction

We have used standardized satisfaction questionnaires for usability evaluation.In order to evaluate the Test Level Satisfaction we have used IBM Computer Usability Satisfaction Questionnaires which contains 19 Questions. The subjects were requested to give a score from 1 to 7 for each of the 19 questions, based on their degree of satisfaction.(strongly disagree - 1 to strongly agree - 7) Secure CodeCity with Evolution's mean CSUQ score is 88.2357 (standard deviation of 6.4507), and it is 11.0638 higher than the that of control group mean score of 77.1719 (standard deviation of 9.345), it evidently showed the acceptance of Secure CodeCity with Evolution over baseline tool, SonarQube, which was used for the evaluation.

By comparing all the metrics it clearly indicates that the Overall usability of the Secure CodeCity with Evolution is far more greater than SonarQube.

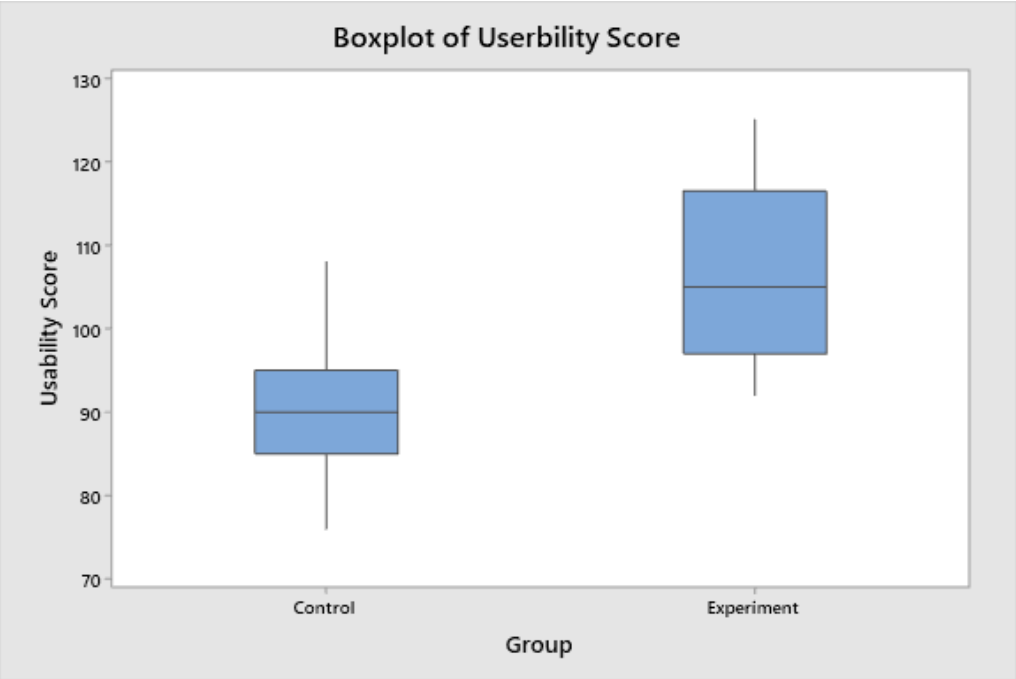


Figure 5.5 Boxplot of overall Usability Score between Experimental and Control Group

5.4.3 Overall correctness of the tasks

A simple rating mechanism has been used to obtain the correctness values for each task. If the answer to a particular task is correct (i.e., the perfect match of the obtained values) the participant was given one point. Likewise, for the mentioned eleven tasks, a maximum score of 24 points could be obtained if all of them were answered correctly. Similarly, 0 marks allocated for the wrong answers and timeouts.

Secure CodeCity with Evolutions's mean correctness score is 16.889 (standard deviation of 1.833) compared with the mean correctness score of 12.182 in control group (standard deviation of 3.027), Box plot is shown in Figure 5.6 for the overall correctness for the both experimental and control groups. Which is denoting considerable overall correctness of Secure CodeCity with Evolutions over SonarQube.

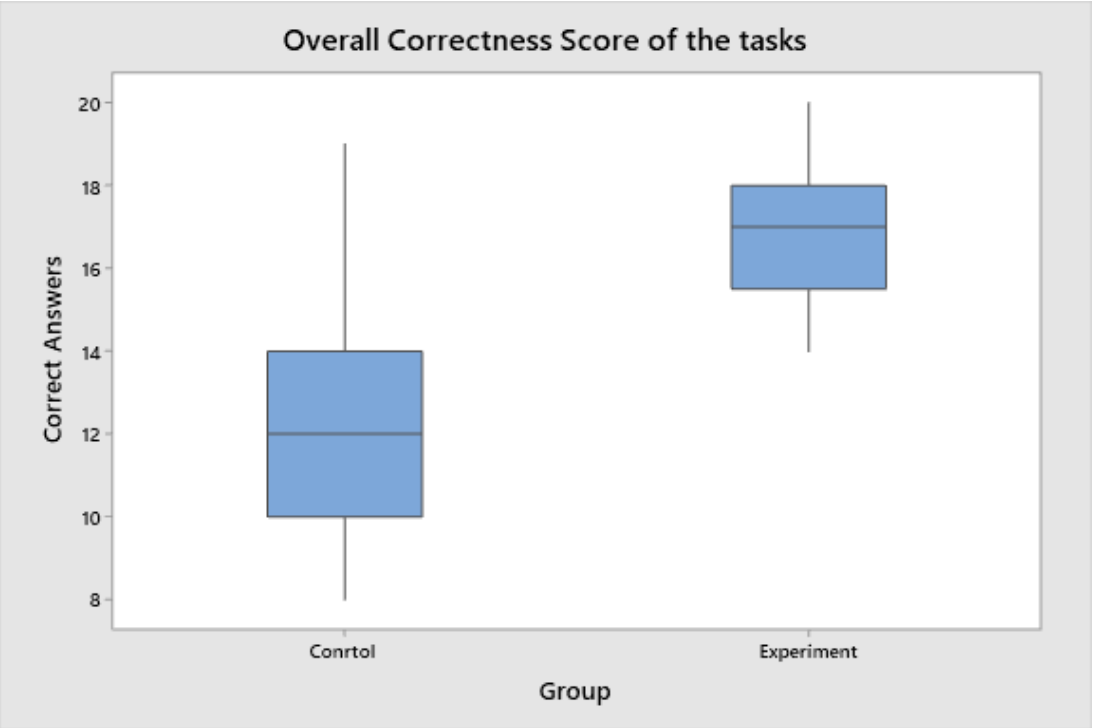


Figure 5.6 Boxplot of overall Correctness Score Time between Experimental and Control Group

5.4.5 Statistical Analysis for hypothesis testing

5.4.5.1 One Way Anova Experimental Design

Variance analysis is used to test whether the experiment result is different from the control

One way Anova experimental design was used assuming:

1. The independent variable for the category variables (categorical variable), depending on the variables must be continuous variables (continuous variable)

2. The mother group must be normal distribution (Normal Distribution)

3. Independent event: The sample must be independent variable (Independent variable) → the sample of the first group does not affect the sample of the second group; the sample of the second group does not affect the first group.

4. Variance homogeneity: The variance of the two groups of samples must be equal.

5.4.5.2 Interpretation of the effect plot

According to the two - factor factorial analysis done by Minitab 19 software, the P value was 0.000 at 5 % significance level

P value	>0.05	H_0 is not rejected at 5% significance level
P value	=<0.05	H_0 is rejected at 5% significance level

Table 5.2Explanation of H0 rejection or H0 acceptance due to the P value

Property	P value	Comment
Overall Accuracy	0.001	$H1_0$ rejected
Usability rating	0.000	$H2_0$ rejected
Time taken	0.007	$H3_0$ rejected

Table 5.3 Statistical values

According to the statistical analysis using Minitab 19 software the P- value < 0.05 at 0.05 level of significance when considering Secure Code City with Evolution (Experiment) and SonarQube (Control). It proves that there is a significant difference in the experiment and control which confirms the alternative hypothesis.

5.5 Conclusion

It was observed that Secure Code City with Evolution outperforms SonarQube regarding accuracy, time efficiency, and usability. Any, stakeholder having a conceptual knowledge in the software analysis domain can use secure Code City with Evolution to answer some basic questions, even without having prior programming knowledge or security related knowledge. Just by seeing the colour spectrum and the spread of the colours across the city, any user can identify what the most security vulnerable classes are. This is allowed through three-dimensional graphical visualization mechanisms. Further, both expert and novice researchers can perform various analysis experiments on the Secure Code City with Evolution framework.

Chapter 6: Conclusion and Future Work

In this chapter a conclusion and a summary of the study in relation to its aims and objectives, problem definition and limitations of this current work is given. Furthermore, at the end of the chapter suggestions for further works are discussed as well

6.1 Secure Code City with Evolution Applications

In this thesis a study was conducted to augment software security in existing software visualization techniques, in order to help the programmers in following SDLC as the research problem. To achieve this initially background literature was studied to select a suitable software visualization Afterward “Code City metaphor” is selected as the best software visualization model.

6.2 Further Work

- Secure Code City with Evolution only supports for the applications developed in Java programming language. This can be further enhanced to enable for compatible for other languages

- Visualization of Evolution can be done based different metrices

- Another possible avenue is when identifying the design level threats from source code, this approach can be extended to visualize STRIDE threat categories

List of References

[1] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo, "Securing web application code by static analysis and runtime protection," in Proceedings of the 13th conference on World Wide Web - WWW '04, New York, NY, USA, 2004, p. 40.

[2] Wijesiriwardana, C. and Wimalaratne, P., 2017. On the detection and analysis of software security vulnerabilities. 2017 International Conference on IoT and Application (ICIOT)

[3] B. Liu, L. Shi, Z. Cai, and M. Li, "Software Vulnerability Discovery Techniques: A Survey," in 2012 Fourth International Conference on Multimedia Information Networking and Security, Nanjing, China, 2012, pp. 152–156.

[4] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)," IEEE Symp. Secur. Priv., p. 6, 2006.

[5] B. Chess and G. McGraw, "Static analysis for security," IEEE Secur. Priv. Mag., vol. 2, no. 6, pp. 76–79, Nov. 2004.

[6] R. Wettel and M. Lanza, "Visualizing Software Systems as Cities," in 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, Banff, AB, Canada, 2007, pp. 92–99.

[7] A. Aoki et al., "A case study of the evolution of Jun: an object-oriented open-source 3D multimedia library," in Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, Toronto, Ont., Canada, 2001, pp. 524–533.

[8] L. Voinea, A. Telea, and J. J. van Wijk, "CVSscan: visualization of code evolution," in Proceedings of the 2005 ACM symposium on Software visualization - SoftVis '05, St. Louis, Missouri, 2005, p. 47.

[9] S. P. Reiss, "The Paradox of Software Visualization," in 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, Budapest, Hungary, 2005, pp. 1–5.

[10] Information security handbook: develop a threat model and incident response strategy to build a strong information security framework Darren Death - Packt Publishing - 20174

[11] E. Alikhashashneh, R. Raje, and J. Hill, "Using Software Engineering Metrics to Evaluate the Quality of Static Code Analysis Tools," in 2018 1st International Conference on Data Intelligence and Security (ICDIS), South Padre Island, TX, 2018, pp. 65–72.

[12] S. Mohajer Naraghi, "An Improved Method of Static Code Analysis Based on the Context-Sensitive Rules", Majlesi Journal of Mechatronic Systems, vol. 8, no. 3, Sep. 2019.

[13] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in Proceedings of the 11th Working Conference on Mining Software Repositories, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 192–201.

[14] S. Pearson, and A. Benameur, "Privacy, security and trust issues arising from cloud computing", in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pp. 693-702, 2010

[15] WannaCry Ransomware Attack. https: //en.wikipedia.org/wiki/WannaCry_ ransomware_attack.

[16] NEWMAN, L. H. Equifax officially has no excuse. Wired, 2017. https://www.wired.com/story/ equifax-breach-no-excuse/.

[17]M. D'Ambros and M. Lanza, "Software bugs and evolution: a visual approach to uncover their relationship," *Conference on Software Maintenance and Reengineering (CSMR'06)*, Bari, 2006, pp. 10 pp.-238.

[18]A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, Chicago, IL, 2004, pp. 284-293.

[19]S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, Orlando, FL, USA, 2002, pp. 291-301.

[20]J. Lin, Q. Zhang, H. Bannazadeh and A. Leon-Garcia, "Automated anomaly detection and root cause analysis in virtualized cloud infrastructures," *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, Istanbul, 2016, pp. 550-556.

[23] McCormick, B., DeFanti, T., Brown, M. and Zaritsky, R. (1987). *Visualization in scientific computing*. [New York, N.Y.]: ACM SIGGRAPH.

[24] C. Knight and M. Munro, "Virtual but Visible Software," Proc. Fourth IEEE Int'l Conf. Information Visualization, pp. 198-205, 2000.

[25] G. Roman and K. Cox, "Program Visualization: The Art of Mapping Programs to Pictures," Proc. 14th ACM Int'l Conf. Software Eng., pp. 412-420, 1992.

[26] Owasp.org. (2019). *Category:OWASP Top Ten 2017 Project - OWASP*. [online] Available at: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project [Accessed 25 Nov. 2019].

[27] B. Price, R. Baecker, and I. Small, "A Principled Taxonomy of Software Visualization," J. Visual Languages and Computing, vol. 4, no. 3, pp. 211-266, 1993.

[28] G. Roman et al., "A Taxonomy of Program Visualization Systems," Computer, vol. 26, no, 12, pp. 11-24, Dec. 1993.

[29] C. Knight, "System and Software Visualisation," Handbook of Software Engineering and Knowledge Engineering, pp. 1-17, World Scientific Publishing Company, 2001.

[30] J. Maletic, A. Marcus, and M. Collard, "A Task Oriented View of Software Visualization," Proc. First Int'l Workshop Visualizing Software for Understanding and Analysis, pp. 32-40, 2002.

[31]En.wikipedia.org.(2019).*Software visualization*. [online] Available at: https://en.wikipedia.org/wiki/Software_visualization [Accessed 25 Nov. 2019].

[32] P. Caserta and O. Zendra, "Visualization of the Static Aspects of Software: A Survey," IEEE Trans. Vis. Comput. Graph., vol. 17, no. 7, pp. 913–933, Jul. 2011.

[33]M. Fischer and H. Gall, “Evograph: A Lightweight Approach to Evolutionary and Structural Analysis of Large Software Systems,” Proc. 13th Working Conf. Reverse Eng., pp. 179-188, 2006.

[34]Gračanin, D., Matković, K. and Eltoweissy, M., 2005. Software visualization. Innovations in Systems and Software Engineering, 1(2), pp.221-230.

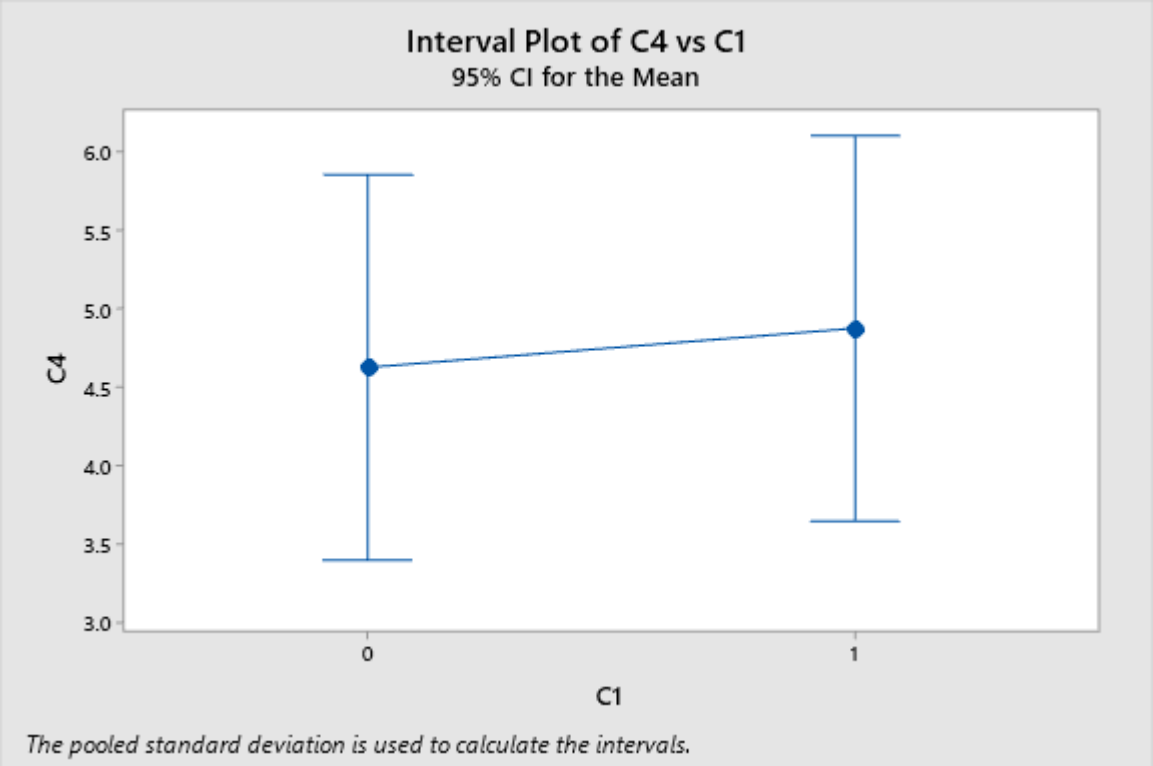
[35] G. McGraw, “Automated Code Review Tools for Security,” Computer, vol. 41, no. 12, pp. 108–111, Dec. 2008.

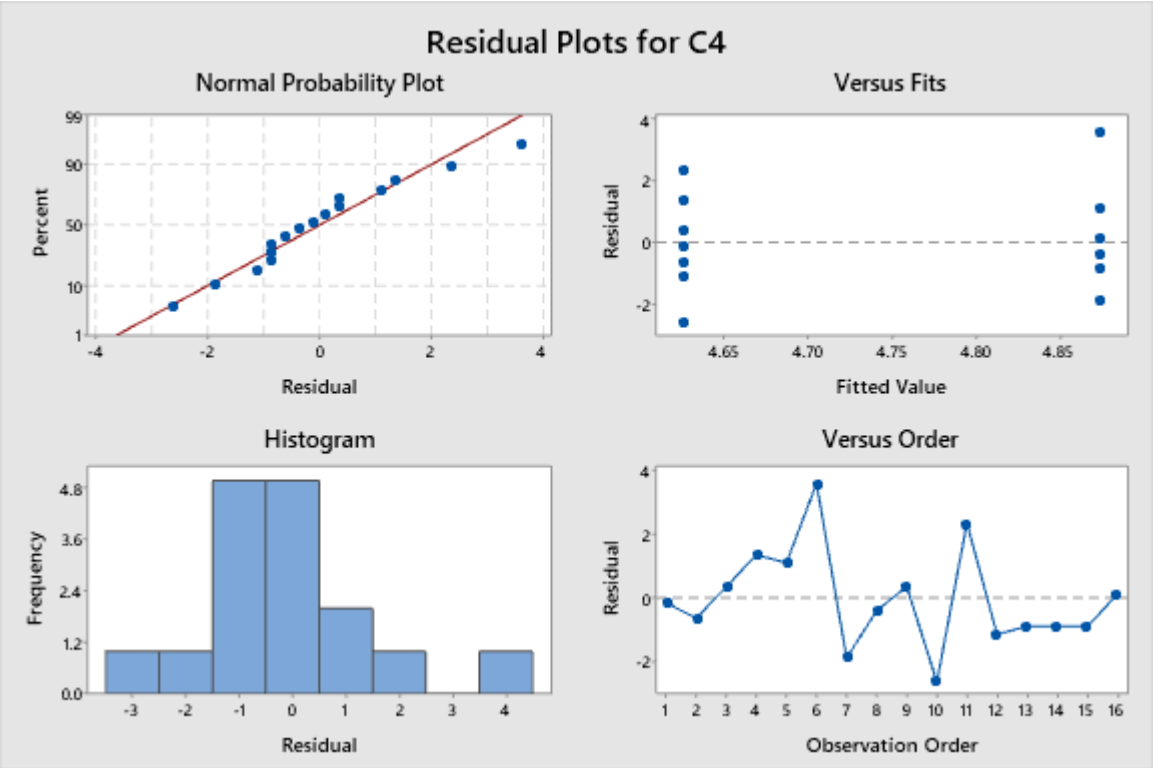
[36] Automotive Cyber Security Market | Industry Report, 2019-2025, 2020

[37] Statista. 2020. Cyber Crime: Reported Damage To The IC3 2019 | Statista. [online] Available at: <https://www.statista.com/statistics/267132/total-damage-caused-by-by-cyber-crime-in-the-us/> [Accessed 15 November 2020].

[38] Owasp.org. 2020. [online] Available at: <https://owasp.org/www-pdf-archive/Cigital_Top10_DallasOWASP-062116.pdf> [Accessed 15 November 2020].

Appendices


Interval Plot of C4 vs C1
95% CI for the Mean
6.0
5.5
5.0
4.5
4.0
3.5
3.0
C4
0
1
C1
The pooled standard deviation is used to calculate the intervals.



One-way ANOVA: C4 versus C1

Method

Null hypothesis	All means are equal
Alternative hypothesis	Not all means are equal
Significance level	α = 0.05

Equal variances were assumed for the analysis.

Factor Information

Factor	Levels	Values
C1	2	0, 1

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
C1	1	382033	382033	16.31	0.001
Error	22	515275	23422		
Total	23	897308			

Model Summary

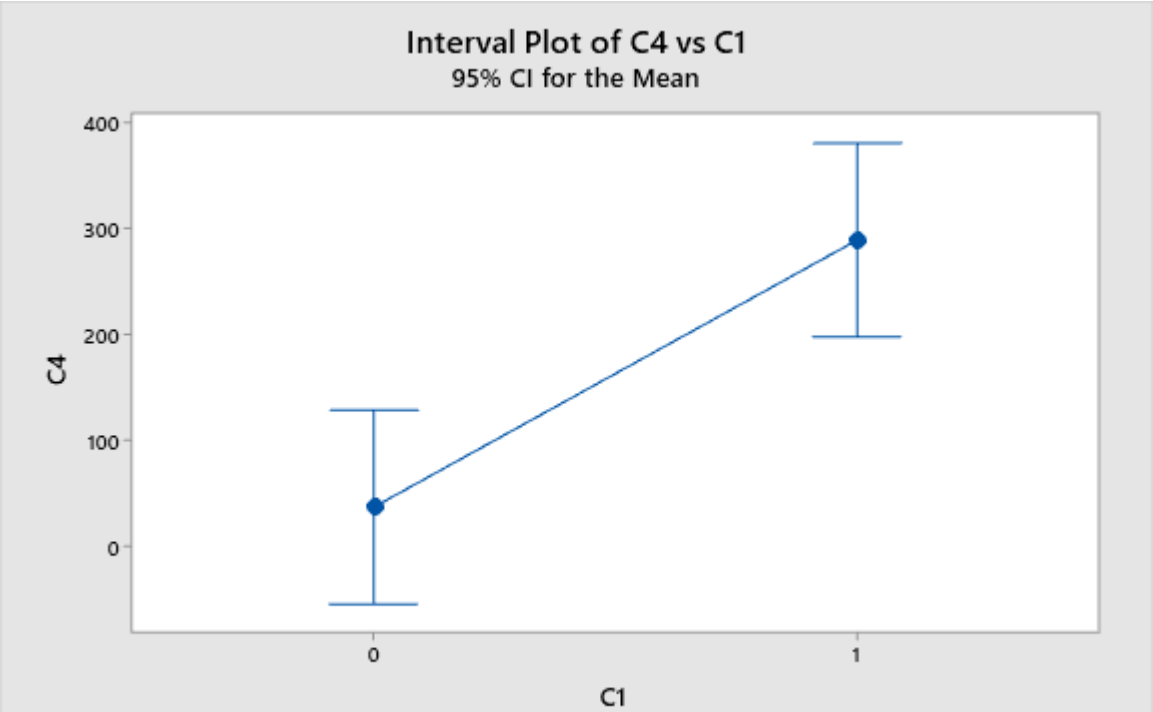
S	R-sq	R-sq(adj)	R-sq(pred)
153.041	42.58%	39.97%	31.66%

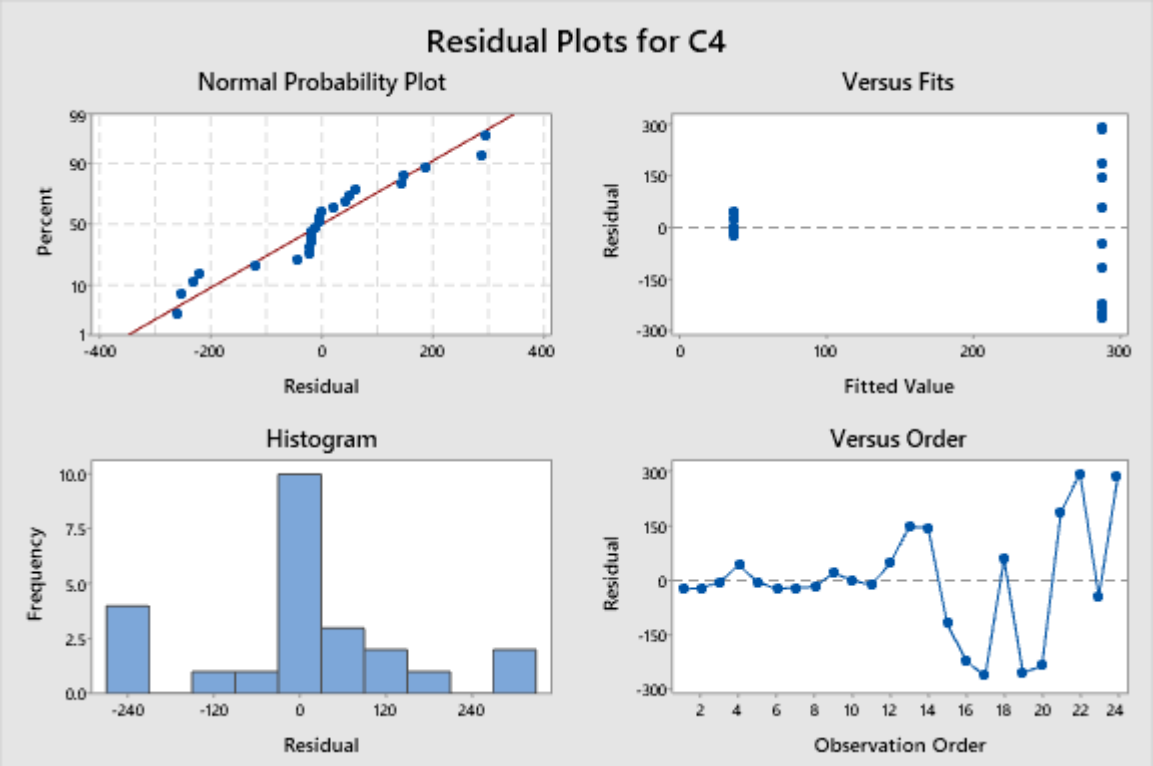
Means

C1	N	Mean	StDev	95% CI
0	12	36.52	25.35	(-55.11, 128.14)

1 12 288.8 214.9 (197.2, 380.5)

Pooled StDev = 153.041





One-way ANOVA: C4 versus C1

Method

Null hypothesis	All means are equal

Alternative hypothesis Not all means are equal
Significance level α = 0.05

Equal variances were assumed for the analysis.

Factor Information

Factor	Levels	Values
C1	2	0, 1

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
C1	1	145.80	145.800	38.94	0.000
Error	18	67.40	3.744		
Total	19	213.20			

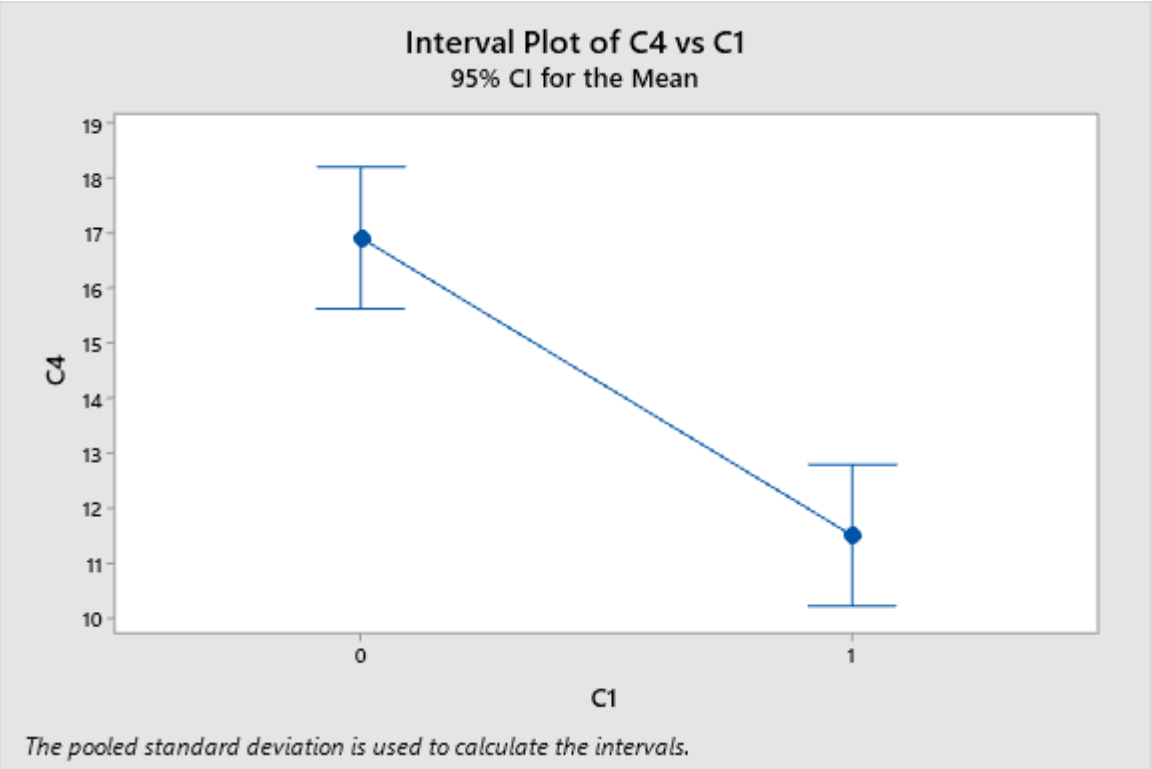
Model Summary

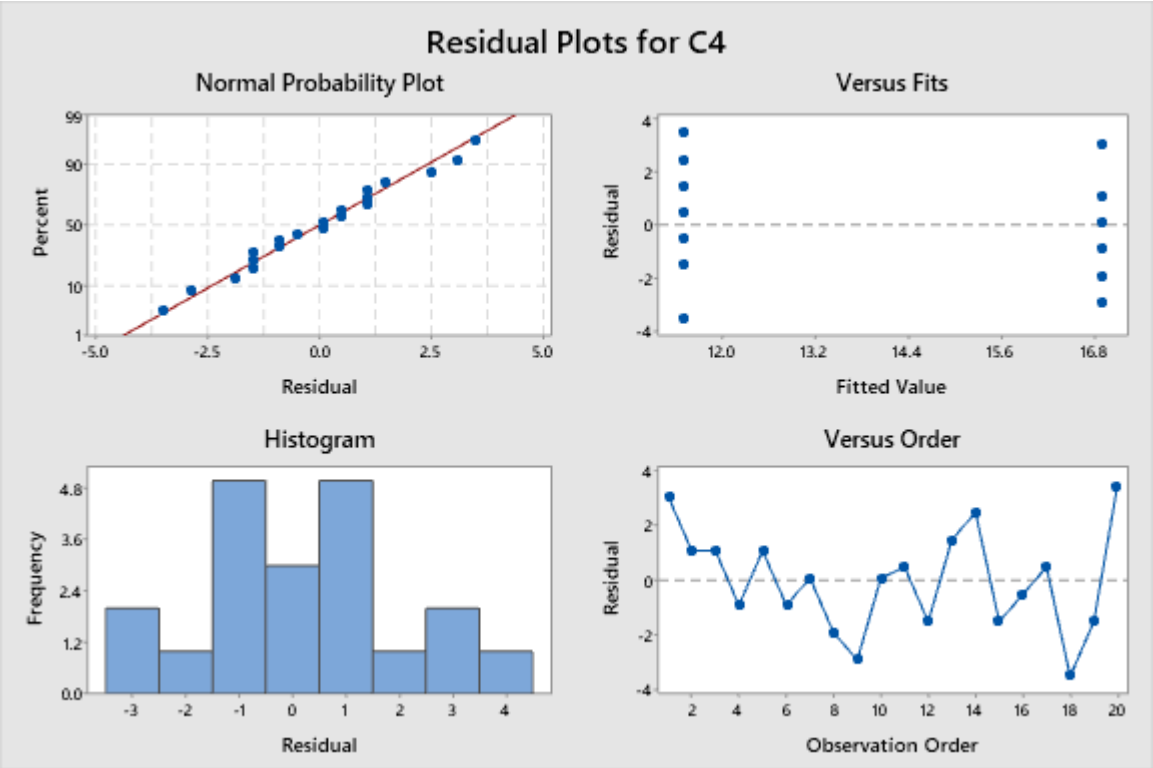
S	R-sq	R-sq(adj)	R-sq(pred)
1.93506	68.39%	66.63%	60.97%

Means

C1	N	Mean	StDev	95% CI
0	10	16.900	1.729	(15.614, 18.186)
1	10	11.500	2.121	(10.214, 12.786)

Pooled StDev = 1.93506





One-way ANOVA: C4 versus C1

Method

Null hypothesis	All means are equal
Alternative hypothesis	Not all means are equal
Significance level	α = 0.05

Equal variances were assumed for the analysis.

Factor Information

Factor	Levels	Values
C1	2	0, 1

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
C1	1	1080	1080.5	9.40	0.007
Error	18	2069	115.0		
Total	19	3150			

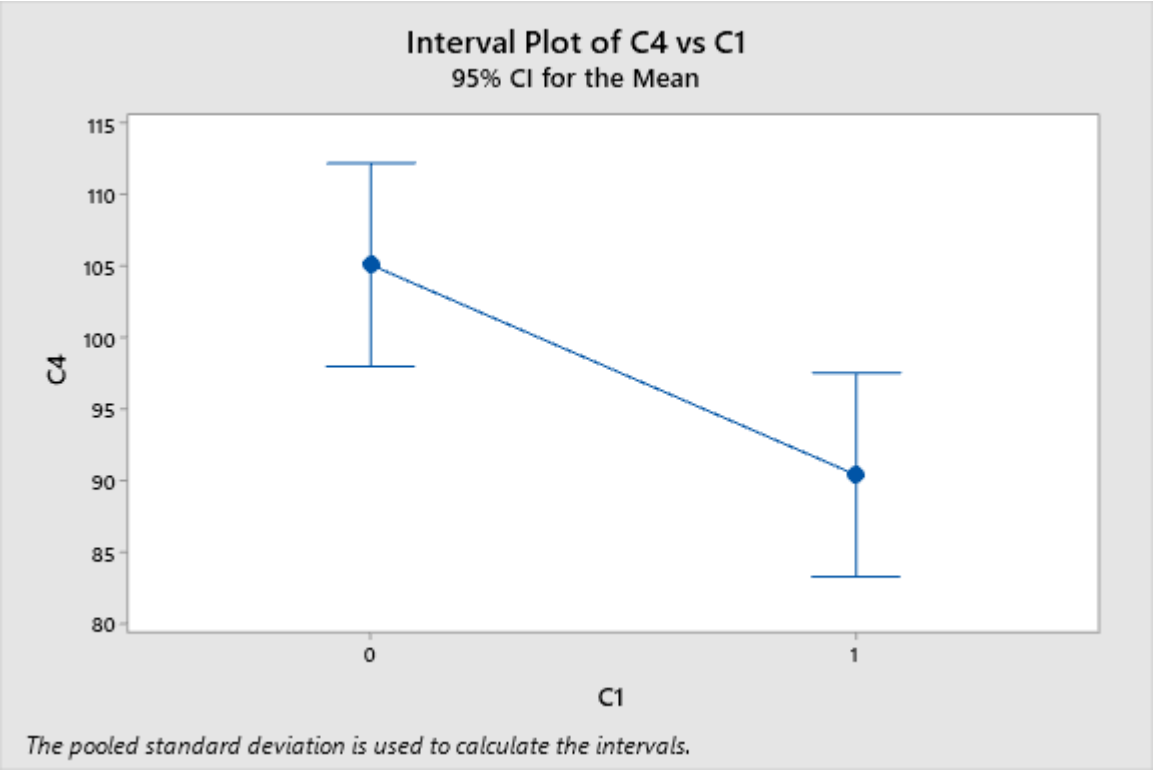
Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
10.7220	34.30%	30.65%	18.89%

Means

C1	N	Mean	StDev	95% CI
0	10	105.10	11.04	(97.98, 112.22)
1	10	90.40	10.39	(83.28, 97.52)

Pooled StDev = 10.7220



The pooled standard deviation is used to calculate the intervals.