Secure CodeCity
A Framework For Security Vulnerability Visualization

A.A.T.G.Abeysinghe
14000016
M.A.S.Shalika
14001357
M.S.N.Ahamed
14000042
S.M.Mufarrij
14000921
Supervisor: Dr. Prasad Wimalaratne
Co-Supervisor: Mr. Chaman Wijesiriwardana

Submitted in partial fulfillment of the requirements of the
B.Sc (Hons) in Software Engineering 4th year Project (SCS 4123)
January 14, 2019

Declaration

We certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. We also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name:

………………………………………………

Signature of Candidate Date:

Candidate Name:

………………………………………………

Signature of Candidate Date:

Candidate Name:

………………………………………………

Signature of Candidate Date:

Candidate Name:

………………………………………………

Signature of Candidate Date:

This is to certify that this dissertation is based on the work of Ms. A.A.T.G.Abeysinghe, Ms.M.A.S.Shalika, Mr. M.S.N.Ahamed, and Mr. S.M.Mufarrij under my supervision. The dissertation has been prepared according to the format stipulated and is of the acceptable standard.

Supervisor Name: Dr. Prasad Wimalaratne

………………………………………………

Signature of Supervisor Date:

This is to certify that this dissertation is based on the work of Ms. A.A.T.G.Abeysinghe, M.A.S.Shalika, Mr. M.S.N.Ahamed, and Mr. S.M.Mufarrij under my supervision. The dissertation has been prepared according to the format stipulated and is of the acceptable standard.

Co-Supervisor Name: Mr. Chaman Wijesiriwardana

………………………………………………

Signature of Co-Supervisor Date:

Abstract

The conventional practice in the software industry towards the resolution of security issues, is to discover those issues during the testing phase of the Software Development Life Cycle and to implement software patches in order to conceal those issues.Such resolutions have resulted in an upsurge of effort and resources,while being unable to eliminate the root symptoms of the security issues. By obliterating the pitfalls of the above approach,an approach which provides a significant focus on integrating Security with each phase of the software development process has been emerged.This approach is named as Secure Software Development Approach, and is known to be leading to the development of more secure and reliable systems.On the other hand,the fundamental idea behind software visualization is to create visual interfaces in order to help developers in understanding different aspects of a source code.Software Visualization has currently become a major topic in the world of research where a large scale effort to find effective software visualization mechanisms, is undertaken by scientific community.Although Secure Software Development and Software Visualization are sturdy approaches inimitably, Secure Software Development does not incorporate intensive software visualization mechanisms such as metaphors in order to manifest the critical information of security issues in software.The purpose of this research is to bridge the aforementioned gap by introducing software visualization to software security.

The research was commenced by analyzing the existing visualization models, for the visualization of security issues in a source code. It was discovered that the ‘CodeCity’ model can be well aligned with the purpose of security vulnerability visualization. Therefore ‘CodeCity’ was selected as the software visualization approach in visualizing security vulnerability information of a particular source code.The research was conducted based on OWASP security vulnerability categorization and related countermeasures. To embrace the aforementioned purpose, a novel framework named ‘Secure CodeCity’ was proposed.

The resulting “Secure CodeCity” visualization focuses on providing a structural overview of the software system, while unveiling security vulnerability information in each levels of the software projects in an attractive and effective manner. This solution proposes several functionalities which will assist programmers in resolving software security issues while following the Secure Software Development approach.

Acknowledgement

This thesis is the result of nearly one year of devoted work by which we were fortunate to have the unconditional assistance of several people who have been extremely supportive in various ways. First and foremost we would like to offer our humble gratitude to Dr. Prasad Wimalaratne, our supervisor, for been extremely supportive and for the guidance he had given us. He had been a great listener, a great advisor, and a great teacher.

We are also grateful to Mr. Chaman Wijesiriwardana, the co-supervisor, a lecturer at the University of Moratuwa, Mr.Kavinda Bandara Herath, Technical lead at Codegen International and Mr. Lahiru Wijesekara, Senior Software Engineer at Epic Lanka for the tremendous encouragement, support and the guidance given throughout this project.

Our deepest gratitude goes to our beloved parents for their unconditional support, love and encouragement extended towards us throughout all the ups and downs of our life. Finally, we would like to thank all the people who had helped us throughout this project.

List of Acronyms

SDLC - Software Development Lifecycle
SSDLC - Secure Software Development Lifecycle
OWASP - Open Web Application Security Project
DFD - Data Flow Diagram

US - United States
UML - Unified Modeling Language
SQL - Structured Query Language
IDE - Integrated Development Environment
XML - Extensible Markup Language
API - Application Program Interface
JSON - JavaScript Object Notation
UCSC - University of Colombo School Of Computing
CI/CD - Continuous Integration/Continuous Deployment

Table of Contents

List of Figures

List of Tables

Chapter 1 : Introduction

The exponential growth of software consumption has raised many new challenges in the domain of security. One such challenge is securing software systems.The primary reason behind securing the software systems is that the software systems are comprised of confidential and sensitive information such as Personally Identifiable Information (PII) [4]. In securing the software systems, conventional approaches to security are operating system security, antivirus and firewalls [1, 4].*Application Security* is another aspect of securing software systems which is based primarily on finding and fixing known security issues after they have been exploited by hackers in the fielded systems [1]. Since Application Security follows naturally from a networkcentric approach to security by embracing standard approaches, such as penetration and patching, and input filtering, it is not adequate to secure the software [1]. Hence, Software Security which is a type of computer security aims to address the aforementioned weaknesses directly by focusing on a secure design and implementation of software [1].

Software defects can occur in a software's design and its implementation where it can be categorized as 'design flaws' and 'implementation bugs' [1, 2, 5]. An implementation bug is a defect in the implementation level [1, 2, 5] and a flaw is a design-level or architecture-level software defect [1, 2, 5].Security breaches begin by exploiting a vulnerability. A vulnerability is a weakness in a system, application, or network that is subject to exploitation or misuse [8]. In the context of software security, security vulnerabilities are security-relevant software defects that can be exploited to cause an undesired behavior [2]. Hence, to fortify software security, it is important to ensure that the   security bugs are eliminated and/or are made harder to exploit [1].

The fundamental concept behind software visualization is that by creating visual interfaces, creators can help developers and others to understand important information regarding the code. A lot of the power of software visualization has understanding relationships between pieces of code, and presenting this information in an accessible way. This dissertation proposes an approach to visualize the security vulnerabilities of a particular software application by using a three dimensional graphical metaphor, and to suggest appropriate countermeasures to rectify the identified defects.The proposing solution uses static code analysis to identify the security vulnerabilities with respect to *OWASP* vulnerability types.

1.1 Motivation

Security vulnerabilities are not restricted to a few products but affect vendors and products available on the software production market [1]. A significant amount of software defects arise due to implementation bugs and architectural flaws [1, 5, 7]. The consequences of a software malfunction or a security breach might lead to a recall, millions in lost revenue or a loss of sensitive customer data [7]. The current challenge that software companies come across is to maintain the software quality with security while accelerating innovation [9]. Due to the necessity of security in a software application, security factor has been added as a characteristic rather than a sub-characteristic in ISO/IEC 25010:2011 Software Quality characteristics [10].

In contrast to the traditional approach of software development, in which the security is concerned only at the final phases and therefore causes high effort and cost for resolution of issues[1, 4], focusing on security during the early stages is proven to be having major reductions in operational vulnerabilities, resulting reductions in software patching. Building Security In paradigm introduced by Gary McGraw in 2004 [1] and Microsoft Trustworthy Initiative [12] introduce a Secure Software Development Lifecycle (Secure SDLC) which implies that security should be built in along with the development of the software by integrating into all phases of the SDLC [1, 12]. A case study showed that the cost to fix requirement problems identified later in the project cost close to $2.5 million while the cost to fix these problems early in the life cycle was $0.5 million. Thus, all these facts have driven to look for new ways in developing software over standard software development processes to further reduce overall software risk,by integrating security into all phases of the SDLC[1, 12].

The general focus of all the practices is to establish a set of practices in order to move developers into a Secure SDLC. This aimed at reducing the number and severity of security vulnerabilities in software and hence identify and manage the security issues throughout the development instead of only at testing and maintenance.

Finding solutions to contribute the aforementioned Secure Software Development is an active research area which leads to immense advantages related to software security.Therefore the main motivation of this research was to contribute Secure Software Development by coming up with a brand new approach which introduces software visualization to software security.

1.2 Problem Definition

Software security is an idea implemented to protect software against malicious attacks and other hacker risks so that the software continues to function correctly under such potential risks.[17] Software security is necessary to provide integrity, authentication and availability of a software.

A significant amount of software defects arises due to implementation bugs in software. The consequences of a software malfunction or a security breach might lead to a recall, millions in lost revenue or a loss of sensitive customer data. According to [18], A report published by Software Engineering institute of Carnegie Mellon University estimates that the Cost of Data Breach in 2016 is 158 dollars per record across the globe, and in particular this amount is 221 dollars per record in US. According to the 2016 Breach Level Index report [19], there were 974 reported incidents of data breaches, with 728 of them occurring in the United States alone, resulting in the loss of millions of confidential documents. Since the universality of software in every field of the world, more companies are focusing on the legal and the financial pressure to assure the security of their software applications in need of securing data. It is crystal clear that ensuring the security of an application is not a trivial task, and needs special attention.

The Building Security In paradigm introduces seven touchpoints including code review, architectural risk analysis, penetration testing, risk-based security tests, abuse cases, security requirements and security operations to be used by integrating with the SDLC [2]. Touchpoints are process-agnostic software security best practices applied on a software artifact including requirements, use cases, design documents, architecture documents, code, test plans, and test results [2] which are aimed at identifying security issues in different phases of the SDLC. Whenever a security issue in one phase is not resolved, it can be propagated to security ramifications in a succeeding phase. In order to solve the security ramification, the real causes of the issue need to be solved. This shows the importance of augmenting the concerns of software security into each phase of SDLC.

Hence, this dissertation proposes an approach that targets secure software development by visualizing the design flaws in the design during the design phase and security vulnerabilities of the source code during the development/ testing phases, given a project's source code and DFD.The severity of each class in the source code will be visualized using a specific color spectrum, and countermeasures for the detected vulnerabilities will be suggested.

1.3 Aims and Objectives

1.3.1 Aim

To Introduce software security to software visualization by deriving a solution which visualizes a source code and its vulnerability details in the form of a three dimensional graphical metaphor alongside with related countermeasures, in order to make programmer's life easier when following Secure Software Development Life Cycle.

1.3.2 Objectives

- To perform a background study on software security processes related to each stage of SDLC

- To study existing software visualization models and related literature

- To select a suitable visualization model which can be used for software visualization
- To study existing software vulnerability categorizations and related details
- To study existing static code analysis techniques/tools to be used in the code review, and related literature
- To select a suitable software vulnerability categorization and an appropriate static code analysis technique/tool to identify security vulnerabilities in accordance with the selected software vulnerability categorization
- To investigate methods to map the vulnerability indications to the graphical metaphor model and to show the severity levels of the vulnerabilities.
- To Find the ways to get counter measures for the vulnerabilities identified.
- To investigate the possible method to redirect to the IDE to edit the code by user after identifying the vulnerabilities.
- To identify a suitable benchmark project which can be used as an input to the software visualization
- To implement the proposed solution
- Evaluate the implemented solution by using sample/benchmark project(s).
- To compile a thesis detailing the background, research methodology & design, results, and evaluation process.

1.4 Scope

The proposed framework in this dissertation is only focused on analyzing and visualizing security vulnerability related information of Java web-based projects.Because the number of vulnerability types identified by SonarQube is maximum for the Java Web Application projects compared with the other supported languages.It is only focused on software security and not on any other approaches to computer security such as network security. The security vulnerabilities categorized irrespective of OWASP T10 will not be processed form the framework.

1.5 Structure of the Dissertation

The rest of this document is as follows.Chapter 2 reviews the background and the existing literature related to the project.Chapter 3 describes the design architecture of the project in detailed and Chapter 4 presents the implementation of the project. The Chapter 5 illustrates the evaluation results and Chapter 6 concludes the dissertation with a conclusion and a discussion about the future work.

Chapter 2 : Background Study

2.1 Introduction

This section discusses the current research approaches and techniques that have been conducted related to the particular area of the study proposing in this dissertation. An analyzing of the relevant concepts and major tools used for architectural risk analysis and code review which considered as the two important practices in seven touchpoints in Secure SDLC are discussed in detail.

There are some specific aims of background study for proposed approach. At first security specific approaches and practices for general SDLC processes have to be studied. Different types of analysis tools available for code review have to be identified.In order to do proper visualization, possible methods to develop the graphical metaphor model and how to map vulnerabilities indications to the graphical metaphor model should be investigated. Finally compare the advantages and disadvantages of discovered details and analyze them according to the applicability of the proposed framework.

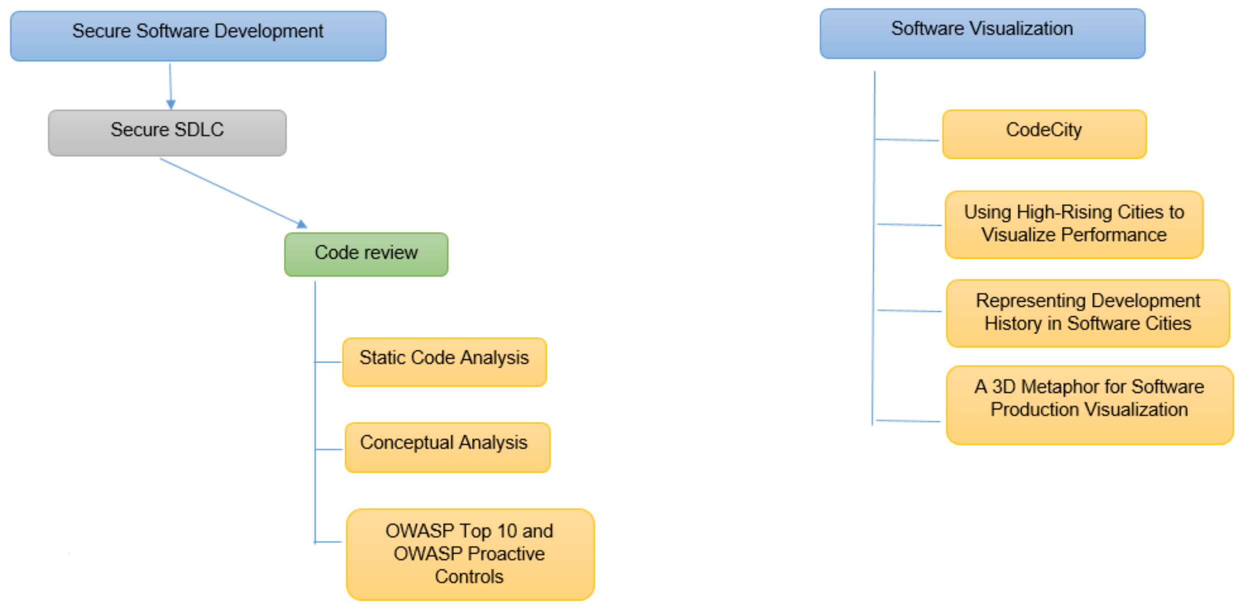

Secure Software Development
Secure SDLC
Code review
Static Code Analysis
Conceptual Analysis
OWASP Top 10 and OWASP Proactive Controls
Software Visualization
CodeCity
Using High-Rising Cities to Visualize Performance
Representing Development History in Software Cities
A 3D Metaphor for Software Production Visualization

Figure 2.1: Hierarchy of Background Study

2.2 Software Visualization approaches

There is a body of work available in the literature focusing on software visualization. These efforts can mostly be put into two categories. The ones which perform 2D visualization and the ones that perform 3D visualization. SeeSoft [43], one of the earliest visualization metaphors, allows one to analyze up to 50,000 lines of code by mapping each line of code into a thin row. Marcus et al. [44] added a new dimension to SeeSoft to support an abstraction mechanism to achieve better representation of higher dimensional data. Goldberg and Robson introduced SmallTalk, one of the first visualized development environment [45].

Recently, there have been more work focusing on 2D visualization. Code Bubbles [1] suggested a collection of editable fragments that represent functions in a class (see Figure[figure number]). Code Gestalt[46] used tag overlay and thematic relations. Lanza and Ducasse [47] proposed categorizing classes and their internal objects into blocks called Blueprints. Gutwenger et al. [48] proposed an approach for improved aesthetic properties of UML diagrams when visualizing hierarchical and non-hierarchical relations. Radfelder and Gogolla [49] extended UMLs by adding third and fourth dimension of data in a way that they can show both dynamic and static aspect of diagram in a single view. Balzer et al. [50] introduced hierarchy-based visualization for software metrics using Voroni Treemaps. Additionally, Holten [51] used both hierarchical and non-hierarchical data to visualize adjacency relations in a software. Hawes et al. [52] presented CodeSurveyor a spatial visualization technique that aims to support code comprehension in large codebases by allowing developers to view large-scale software at all levels of abstraction. The common observation among these efforts is that they are all based on 2D environments and were mostly suitable for expert users.

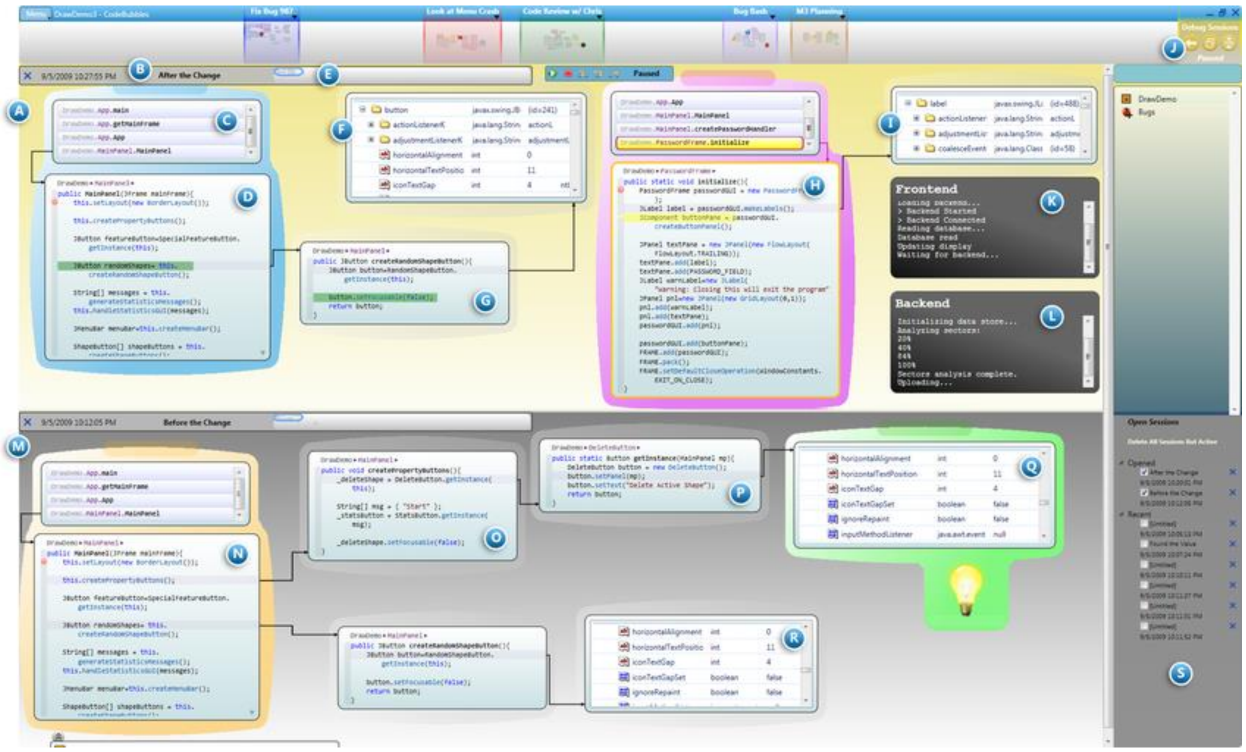


Figure 2.2: Code Bubble: suggested a collection of editable fragments that represent functions in a class [1].

Why prefer 3D over 2D?

3D environments tap into the spatial memory of the user and help with memorizing the position of objects. These objects could be classes or methods. There are also studies which provide evidence that spatial aptitude is a strong predictor of performance with computer-based user interfaces. For instance Cockburn and McKenzie [53] have shown that 3D interfaces that leverage the human's spatial memory result in better performance even though some of their subjects believed that 3D interfaces are less efficient. Robertson et al. [54] have also shown that spatial memory does in fact play a role in 3D virtual environments. As a result, there are a variety of work available in the literature that are focused on 3D software visualization.

2.2.1 CodeCity

The paper published by Richard Wettel and Michele Lanza [39] on Visualizing software systems as cities outlines a 3D visualization approach which gravitates around the city metaphor, an object-oriented software system is represented as a city that can be traversed and interacted with the goal is to give the viewer a sense of locality to ease program comprehension. The key point in conceiving a realistic software city is to map the information about the source code in meaningful ways in order to take the approach beyond beautiful pictures. This included several concepts that contribute to the urban feeling, such as appropriate layouts, topology, and facilities to ease navigation and interaction. It gives some experimented results from their approach on a number of systems.

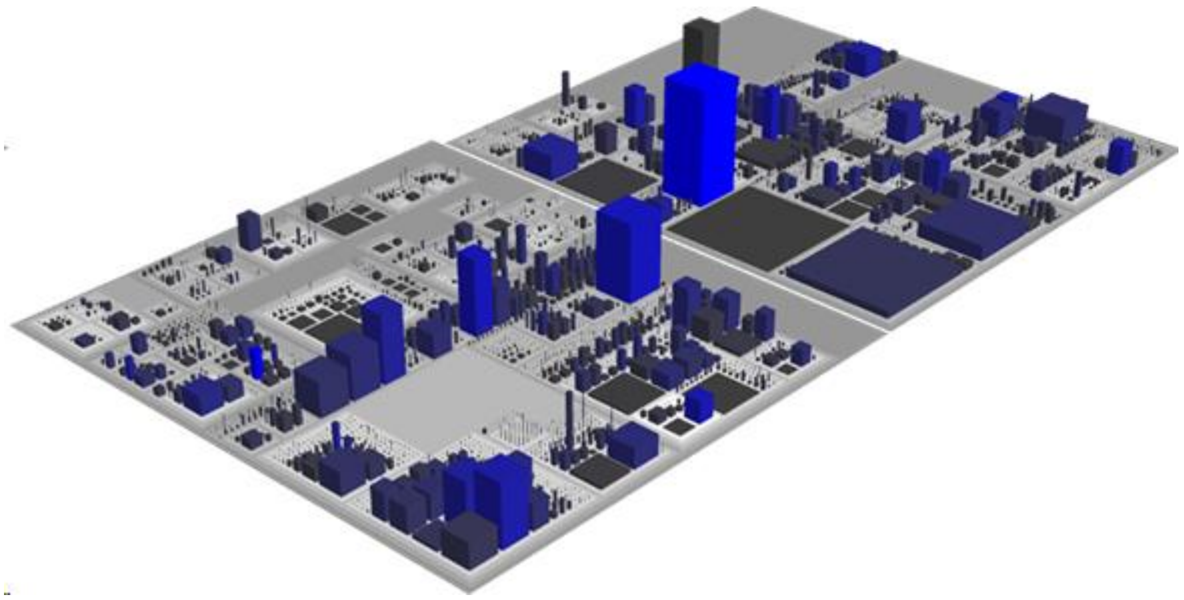


Figure 2.3 : CodeCity Project

In above figure, the classes are represented as buildings in the city, while the packages are depicted as the districts in which the buildings reside. The visible properties of the city artifacts depict a set of chosen software metrics, as in the polymeric views of Code Crawler.

2.2.2 Using High-Rising Cities to Visualize Performance

In this paper[40], they visualize the performance data from a real-time profiler. We visualize program execution as a three-dimensional (3D) city, representing the structure of the program as artifacts in a city (i.e., classes and packages expressed as buildings and districts) and their program executions expressed as the fluctuating height of artifacts. Through two case studies and using a prototype of our proposed visualization, we demonstrate how our visualization can easily identify performance issues such as a memory leak and compare performance changes between versions of a program.

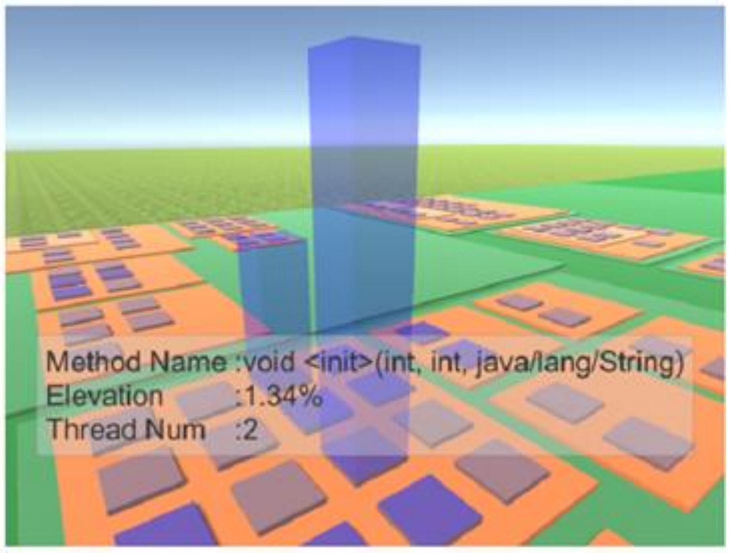


Figure 2.4: Using High-Rising Cities to Visualize Performance in Real-Time

A high-rise building indicates a performance issue. In this example, we find that the high riser is caused by a thread leak with multiple-threading. We find that the method invokes a new thread each time the user restarts for a new game.

2.2.3 Representing Development History in Software Cities

Also CodeCity concept has been used for Representing Development History in Software Cities [41], which describes a systematic approach to utilize the city metaphor for the visualization of large software systems as evolving software cities. The main contribution is a new layout approach which explicitly takes the development history of software systems into account and makes history directly visible in the layouts.

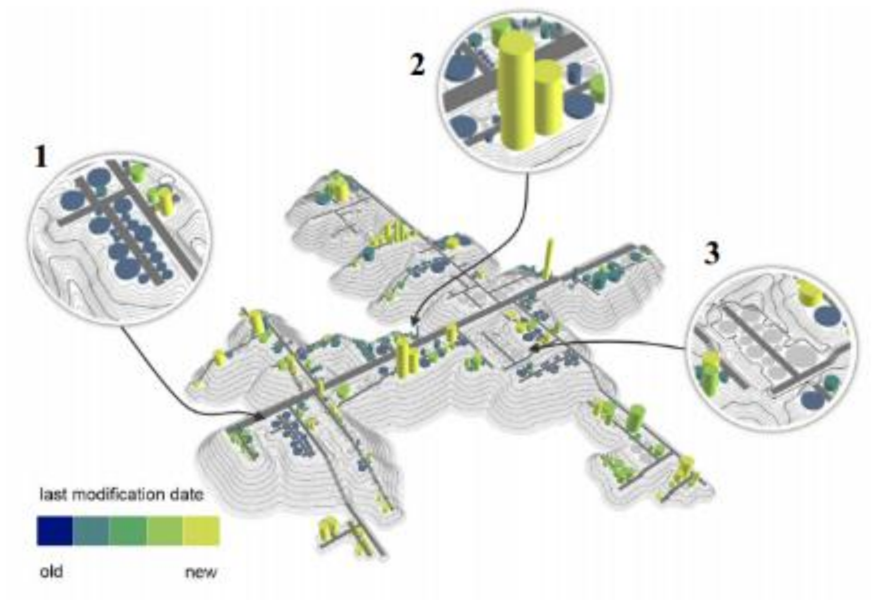


Figure 2.5: Modification history map for CrocoComos

These layouts incrementally evolve in a very smooth and stable way during the development of the represented software system. They are used as a visualization platform for integrating a large variety of product and process data and thus create a coherent set of specialized visualizations.

The authorship map in figure 2.6 shows a different situation with respect to code ownership and the contributions of different developers for the jMonkeyEngine system, a 3D graphics library. Six developers are modifying almost all classes in the system. There seem to be no clear responsibilities of developers for particular parts of the system.

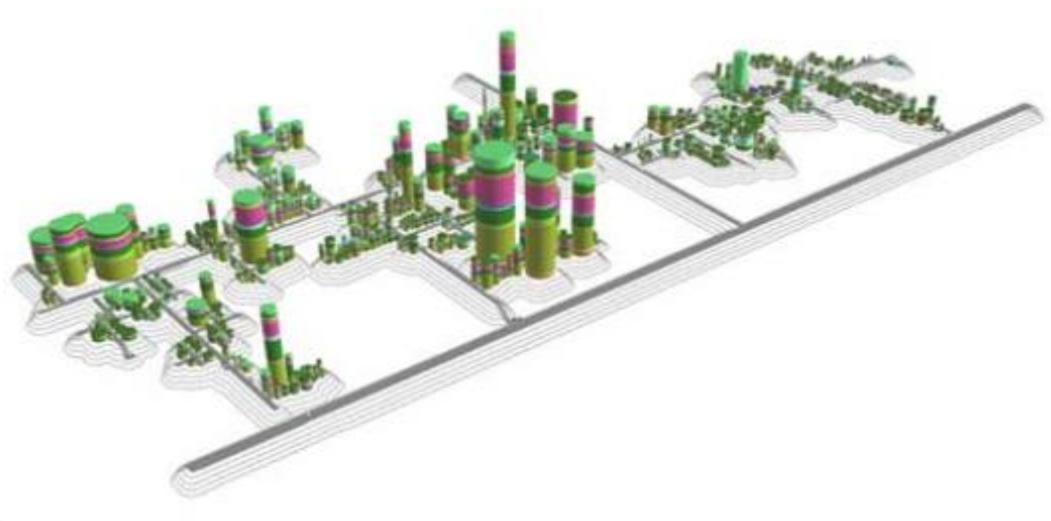


Figure 2.6: Authorship map for jMonkeyEngine

2.2.4 A 3D Metaphor for Software Production Visualization

There is a research paper describing “A 3D Metaphor for Software Production Visualization”. Software development is difficult because software is complex, the software production process is complex and understanding of software systems is a challenge. In this paper they propose a 3D visual approach to depict software production cost related program information to support software maintenance. The information helps us to reduce software maintenance costs, to plan the use of personnel wisely, to appoint experts efficiently and to detect system problems early.

Program visualization can provide a large reduction in effort associated with program understanding. However, in order to increase the reactivity of a development environment with respect to business realities, highly specialized, visual information should be delivered in a timely fashion to those people that can affect greatest impact.

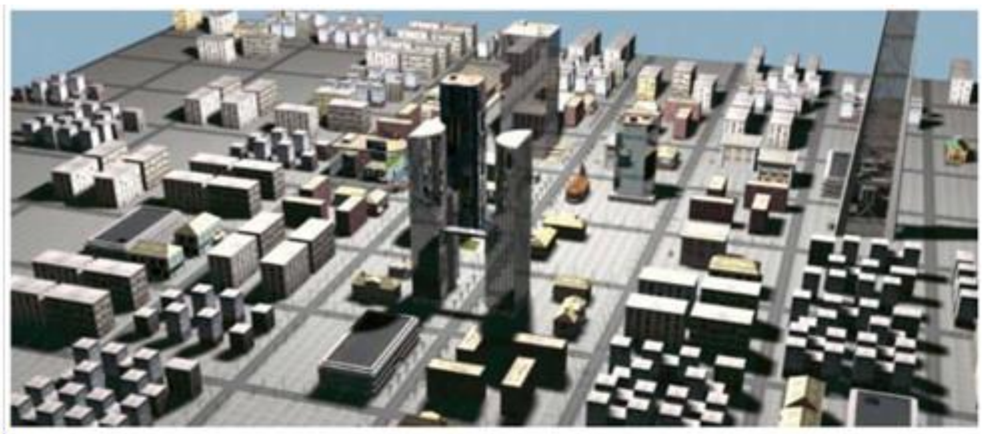


Figure 2.7:3D City from Top without Business Info

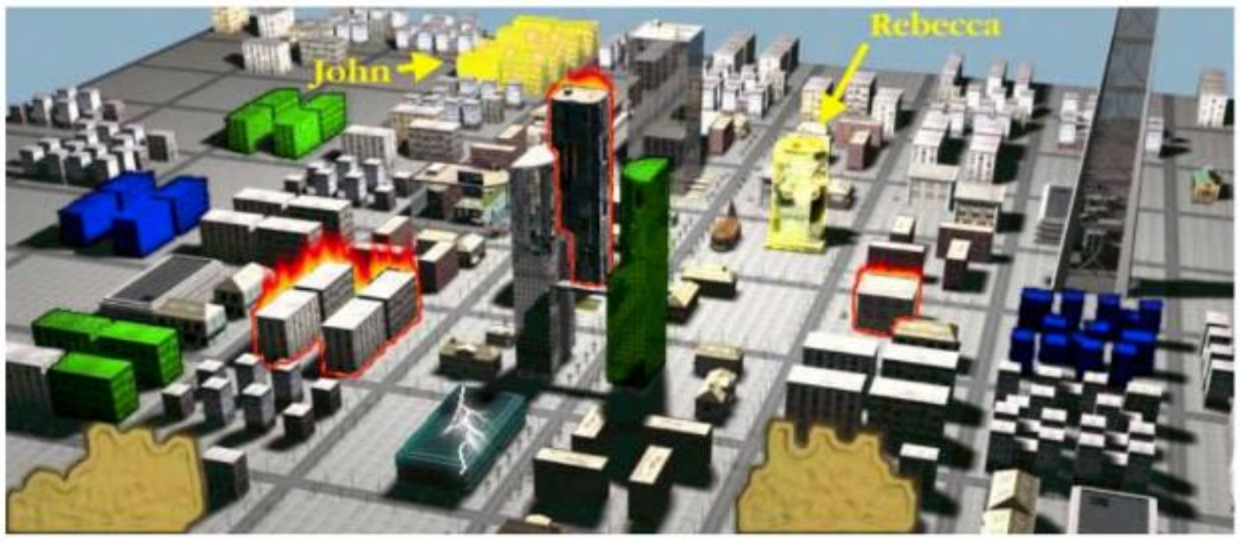


Figure 2.8: 3D City from Top with Business Info

Although, Figure 2.7 helps to visualize various static as well as dynamic aspects of a reverse engineered system and hence increase the understandability of the system, they do not provide support for the various additional demands that developers, designers, vendors and project managers have. Therefore, we apply a cost focused metaphor over the 3D city metaphor, to visualize additional business related information of a system. Figure 2.8 shows approach taken on research.Many more production related issues exist that could be depicted. Vendors and managers might see the quality of a software system at once, by having a short look at the 3D city. Lots of fire, flashes and mud indicate high cost areas of code and unacceptably high risk regarding the ongoing health of the system.

2.3 Secure Software Development Processes

A growing body of research has been conducted in identifying how to integrate security within software development due to the increase in the number of software security problems. As aforementioned in Chapter 1, the pioneer approaches to solving security problems by applying a set of activities through SDLC are Seven Touchpoints introduced in Building Security In paradigm [1] and Security Development Lifecycle (SDL) introduced by Microsoft Trustworthy Initiative [4, 12].

Building Security In [1] is a collaborative effort that provides practices, tools, guidelines, rules, principles, and other resources that software developers, architects, and security practitioners can use to build security into software in every phase of its development. The Building Security In paradigm introduces seven touchpoints as aforementioned in Chapter 1. The touchpoints have been integrated with Software Security knowledge organized into seven knowledge catalogs including Principles, Guidelines, Rules, Attack Patterns, Historical Risks, Vulnerabilities and Exploits as illustrated in Figure 2.1: Mapping of software security knowledge catalogs to various software artifacts and software security best practices [1].

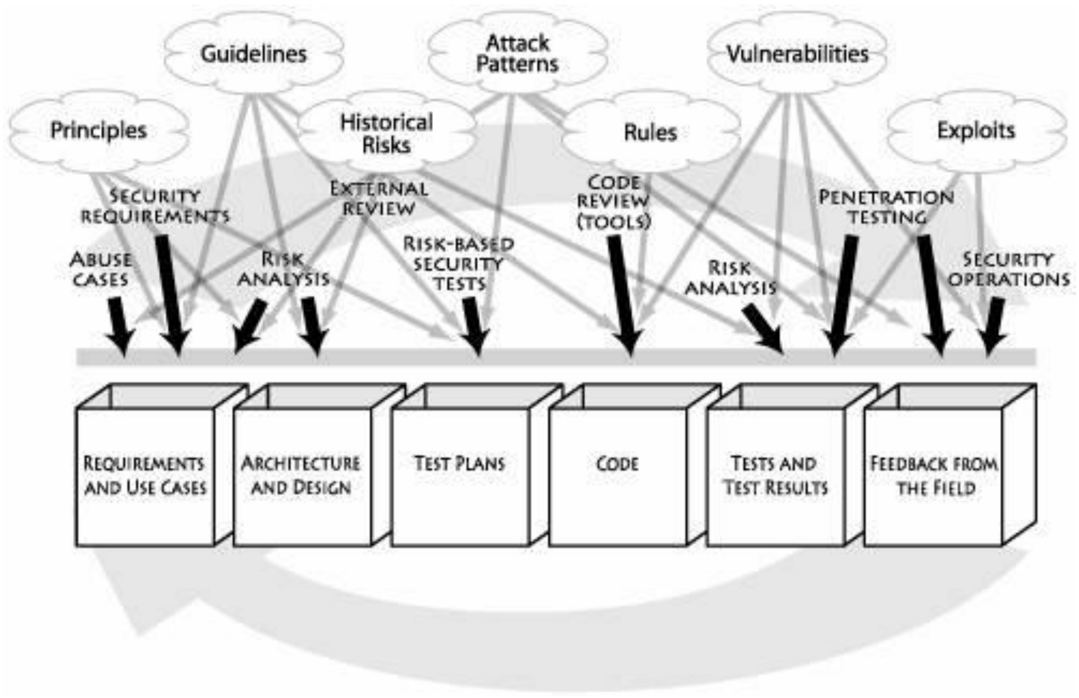


Figure 2.9: Mapping of software security knowledge catalogs to various software artifacts and software security best practices [1]

According to Gary McGraw, software security requires a careful balance by unifying the two sides of attack and defense, exploiting and designing and breaking and building into a coherent whole [1]. In order to make it easier for companies that follow best practices, different touchpoints are in ranking as illustrated in Figure 2.2: Touchpoints numbered according to effectiveness and importance [1].

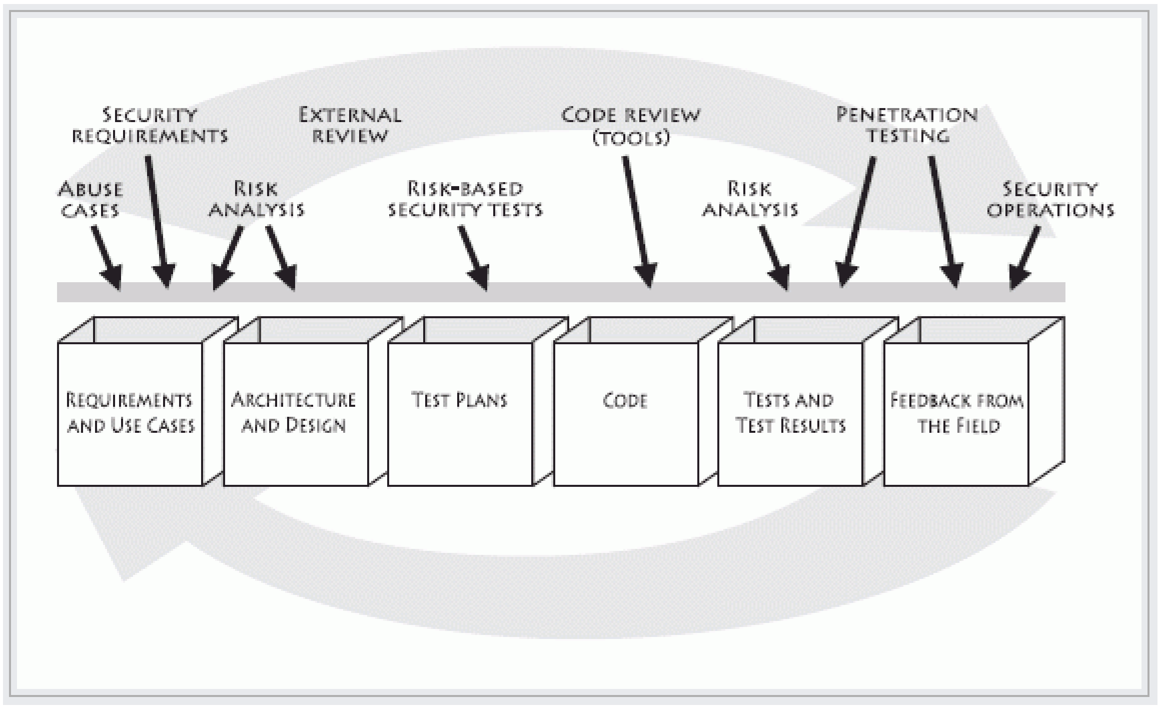


Figure 2.10: Touchpoints numbered according to effectiveness and importance [1]

Despite that Secure SDLC process introduced by Gary McGraw [1] conveys that it follows a traditional waterfall model, the current software development methodologies followed by most of the companies are iterative approaches. This process can be used in iterative approaches where security specific activities can be cycled through more than once as the software evolves. Thus, software security knowledge catalogs can be successfully applied to the SDLC by integrating with touchpoints regardless of the base software development model [1].

Correspondingly Microsoft has carried out a noteworthy effort under its Trustworthy Computing Initiative which focused on people, process, and technology to tackle the software security problem [4, 12]. On the people front, Microsoft trains every developer, tester, and program manager in basic techniques of building secure products. Microsoft's development process has been enhanced to make security a critical factor in design, coding, and testing of every product.
A key part of Microsoft’s Trustworthy Computing is the Security Development

Lifecycle (SDL) [4, 12] which focuses on software development and introduces security and privacy throughout all phases of the software development process as illustrated in Figure 2.3: Microsoft Security Development Lifecycle [4, 12].

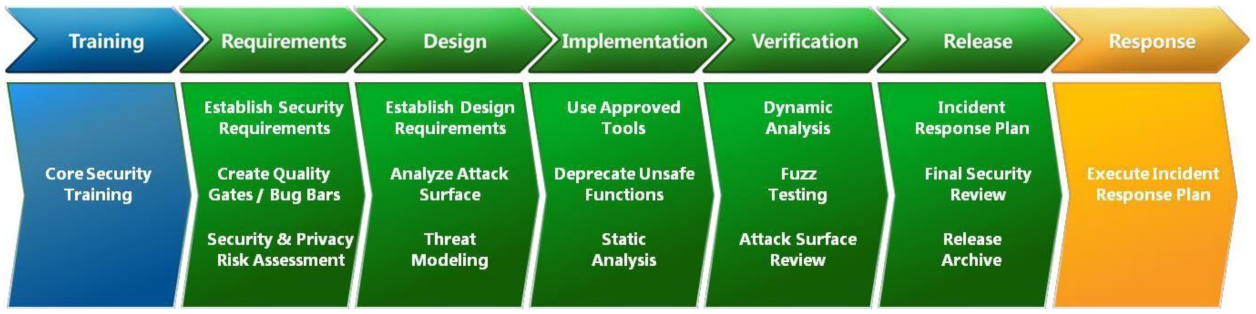


Figure 2.11: Microsoft Security Development Lifecycle [4, 12]

The Microsoft SDL combines a holistic and practical approach to reduce the number and severity of vulnerabilities in Microsoft products [4, 12].

Conforming to the aforementioned approaches introduced to the SDLC, it conveys that Architectural risk analysis and Code review are two significant steps which should be conducted in a security specific SDLC process. The following sections 2.3 and 2.4 include a detailed description of the methodologies followed and tools used in architectural risk analysis and code review respectively.

2.3.1 Code Review

Security Bugs which can be found in the implementation phase of a software project are identified in the code review process. Two approaches to code review have been defined as Dynamic Code analysis and Static Code analysis and this dissertation is only focused on security-specific approaches in static code analysis.

2.3.1.1 Static Code Analysis

Static Code analysis is a software testing method that can be performed in the different stages of the software development to ensure software is free of vulnerabilities introduced to the code due to programming errors [29]. In the context of security review perspective, Static Application Security Testing (SAST) is a well-known method for discovering vulnerabilities and it is classified into a white-box test [30].

The development processes and practices in developing secure software are primarily focused on the use of best practice recommendations which are aimed at addressing common mistakes within a current development process [31]. These include perspective practices such as OWASP Top 10 [38] and Building Security In Meta-Model [1]. A paper published by N. Kaur, et al [32] describes that the efforts in the implementation of secure software have focused on studying implementation vulnerabilities like SQL Injections and Cross-Site-Scripting as listed in OWASP T10.

2.3.1.2 OWASP Top 10 and OWASP Proactive Controls

OWASP Top 10 [38] (hereafter OWASP T10) is the ten most critical web application security risks which provide a powerful awareness document for web application security. The different versions of OWASP T10 are focused on identifying the most common vulnerabilities which have always been organized around risks. It depicts how an attacker can potentially use many different paths through an application to do harm to an organization where each of the paths represents a risk.

OWASP Proactive Controls (hereafter, Proactive controls) [33] is the ten most important control and control categories. This is a developer-centric list of security techniques which can be included in every software project. Each proactive control helps in preventing one or more of the OWASP Top Ten web application security vulnerabilities.

2.3.1.3 Conceptual Analysis

The white paper published by Coverity [34] outlines a practical approach for implementing secure practices into the software development lifecycle. It has introduced a development testing platform which allows the development organizations to coherently integrate code testing into the software development process. Coverity development testing solutions train developers to address both security and quality when testing the code which leads to secure software development practices. The commonly found potentially critical security defects in the source code are identified from this platform and will be provided an aid for the developers to fix them. The major weakness of this platform is the lack of linking the root-cause with the design phase by limiting it to the implementation phase.

The paper published by Sultan S. Alqahtani, et al [35] have proposed, while known vulnerabilities and security concerns are reported in specialized vulnerability databases, these repositories often remain information silos. In this research, a modeling approach is introduced, which eliminates these silos by linking security knowledge with other software artifacts to improve traceability and trust in software products. A Security Vulnerabilities Analysis Framework (SV-AF) is introduced in this approach to support evidence-based vulnerability detection. Two case studies are presented to illustrate the applicability of the presented approach. In these case studies, the National Vulnerability Database (NVD) and the Maven build repository are linked to trace vulnerabilities across repository and project boundaries. In the analysis, 750 Maven project releases are identified as directly affected by known security vulnerabilities and by considering transitive dependencies, an additional 415604 Maven projects can be identified as potentially affected by these vulnerabilities. This approach for ensuring security in a software is limited to the code level and connecting the design phase with the identified bugs is not supported in the framework.

2.3.2 Static Code Analysis Tools

The Coverity Development Testing Platform [34] introduced by Coverity, provides development teams the ability to test code for defects in a non-intrusive manner. It integrates with IDEs like Eclipse or Visual Studio, and developers can identify quality and security defects from within their IDE, without disrupting the development workflow. The identified defects are automatically notified to the developers within the existing workflow, prioritized by risk and impact. Developers have one-click access to a rich defect knowledge base which takes the guesswork out researching unfamiliar defects and helps developers to find the root-cause of a defect in an efficient manner. Considering the fact that many organizations leverage shared code across projects and services, Coverity Static Analysis will also show the development team all of the places across the shared code where that defect exists, so a fix can be applied in all these places. However, this is a commercial tool.

Find-Sec-Bugs [36] is a static analysis tool used to find security audits of java web applications. It can detect 113 different vulnerability types with over 689 unique API signatures. The plugin covers popular frameworks including Spring-MVC, Struts, and Tapestry etc. and available for Eclipse, IntelliJ, Android Studio and NetBeans. Command line integration is available with Ant and Maven. The plugin can be used with systems such as Jenkins and SonarQube and extensive references are given for each bug patterns with reference to OWASP T10 and Common Weakness Enumeration (CWE). A detailed report of the results of the analyzing process is provided from the plugin which can be saved in XML format. A set of predefined bug patterns are available in this tool which has been categorized in accordance with OWASP T10. Despite that, the output reports generated from the tool do not contain the detected bugs as a categorization of OWASP T10.

SonarQube [37] is another static code analysis tool used to collect and analyze source code, measuring quality and providing reports for the project. It combines static and dynamic analysis tools and enables quality to be measured continuously over time. Everything that affects the code base, from minor styling details to critical design errors, is inspected and evaluated by SonarQube, thereby enabling developers to access and track code analysis data ranging from styling errors, potential bugs, and code defects to design inefficiencies, code duplication, lack of test coverage, and excess complexity. The Sonar platform analyzes source code from different aspects and hence it drills down to the source code layer by layer, moving from the module level down to the class level. At each level, SonarQube produces metric values and statistics, revealing problematic areas in the source that require inspection or improvement.

A paper submitted by Harneet Kaur [38] has included a comparison conducted between Find-Sec-Bugs and SonarQube as listed in Table 2-3: A comparison of static code analysis tools (Find-Sec-Bugs and SonarQube) [38].

	Find-Sec-Bugs [31]	SonarQube [37]
Purpose	Finding potential bugs	Managing overall quality assurance
Types of Verification	Code-level design flaws, bad practice, multi-threaded correctness	Bugs, duplications, vulnerabilities, code smell, technical debt, overall quality statistics, and metrics
Plugins and Integration with Jenkins	132 rules written in Java and analyzes Java code only	Customizable 1000+ rules supporting more than 20 languages
Custom Rules	132 rules written in Java and analyzes Java code only	Customizable 1000+ rules supporting more than 20 languages

Analysis Results	Displayed on Jenkins server with no flexibility of customization for false positives	Displayed on SonarQube server with flexibility to eliminate false positives, assign severity levels, close issues and check compliant code examples
Authorization and Accessibility	Non-private accessibility of results on Jenkins server	Only authorized users can access results by logging into SonarQube server

Table 2-1: A comparison of static code analysis tools (Find-Sec-Bugs and SonarQube) [32]

The automated categorization of security bugs into the OWASP T10 categorization is an advantage of the SonarQube [37] tool compared to Find-Sec-Bugs plugin [36]. Despite that fact, the OWASP T10 categorization of the SonarQube is limited to A1, A2, A5, A6, A7, and A9. The inability of generating a final report that can be saved after the analysis is a major drawback of the SonarQube tool. The number of vulnerability types identified for the supported languages by SonarQube tool and the supported OWASP T10 categories is illustrated in the following *Table 2-2: No. of Vulnerability Types identified and supported OWASP T10 categories for each supported languages from SonarQube [37].*

Language	No. of Vulnerability Types	Supported OWASP T10 categories
Java	33	A1, A2, A5, A6, A7, A9
PHP	10	A1, A2, A3, A5
JavaScript	9	A3, A6, A9
C#	6	A6
Flex	6	-
Python	1	-

The Table 2-2 depicts that identification of security bugs by SonarQube is comparatively high for the Java language.

If static code analysis is used at the right stage during the development of a project, it has the capability of identifying critical security vulnerabilities or security bugs which may not appear to the surface during or after the project release [32]. In addition to that, with a tool like SonarQube, the generated false positives in an analysis can be eliminated more efficiently [32]. SonarQube is not only useful for maintaining and assuring the security of one project but the configuration can be used in many projects without the restriction of the language used to develop the project.

2.4 Limitations of current approaches

Software Visualization	Static code analysis tools
Security vulnerabilities are not visualized	Vulnerabilities are not given as level of abstraction
Doesn’t support a second level drilled down view	Output given by static code analysis tool is difficult to refer for fixing, when the project is large

Table 2-3: Limitations of current approaches of Software Visualization and Static code analysis tools

2.5 Summary

The wide variety of software visualization approaches over the last two decades led to a plethora of visualizations, documented in several compendia and classifiable according to several taxonomies. Each visualization targets one or more of the many aspects of a software system and encodes information according to its own visual language. Performing an analysis of several aspects of a software system (e.g., design and evolution) would require conducting separate analyses for each targeted aspects, using a different visualization.
There are metaphors that go beyond words and build on the idea that “a picture is worth a thousand words”. Visualization has the potential of reinforcing metaphors, making them more intuitive and memorable. Therefore, software visualization is a fertile ground for the study of software-related metaphors. Over the last decade, the availability of 3D graphics enabled the appearance of more realistic and easier to grasp visual metaphors, such as landscapes, cities , or solar systems . The proposed framework is based on city metaphors. But none of other city like metaphors provided security related informations.

The aforementioned approaches for software security depict that Static code analysis is conducted in the code review process which is a methodology for software testing used with the aid of static analysis tools focused on identifying security bugs.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 framework from this dissertation has used SonarQube for identifying software bugs and OWASP Top 10 categorization given for the identified software bugs is a major advantage of the tool for the proposed approach for the framework.

Chapter 3 : Design

3.1 Introduction

This chapter describes the proposed design approach to the aforementioned problem in this study with the methodologies used and considerations on designing the solution. Based on the critical review done in the background study, several design concerns were identified. Based on these design concerns and the identified requirements the system architecture was developed.
The system architecture consists of several modules which are explained under this chapter. The detailed descriptions of problem analysis, design constraints and solution approach are explained under respective sections.

3.2 Problem Analysis

The primary aim of this project was to develop a framework to identify security vulnerabilities of software project in implementation phase and represent those vulnerabilities in a attractive usable way by using 3D visualization. To achieve this goal, an exploratory type of research was carried out by exploring relevant documents, dissertations, and tools. The main approach that was used to identify the requirements, functional details and system architecture was by analyzing the information gathered through the background study. The limitations that were identified in the current approaches, ideas and information gathered through concept papers, were incorporated in designing the system architecture.

Considering the fact that security vulnerabilities and architecture-level security flaws are the major causes for security issues as aforementioned in Chapter 1, code review was taken as important role in software development life cycle. The additional reason for selecting the preceding touchpoints was the order of effectiveness of seven touchpoints in the Secure SDLC [1].

Static code analysis tools [36, 37] were explored in order to elect a tool to identify the relevant code-level security bugs as aforementioned in Chapter 2. Using the literature survey conducted, SonarQube [37] were selected as the Static code analysis tool respectively. Owasp Top 10[38] was selected to get countermeasures to vulnerabilities as it links with SonarQube.

However, the inadequacy of a direct approach in inferring the association for the aforementioned problem definition in Chapter 1, the research component was based on discovering an approach to map the detected vulnerabilities to 3D metaphors and provide relevant countermeasures to user.

3.3 Design Constraints and Assumptions

The vulnerability visualization of software project from the Secure codecity Framework can be a complete software application or a component of a software application. According to the background study conducted in Chapter 2, the number of vulnerability types identified by SonarQube is maximum for the Java Web Application projects compared with the other supported languages. Hence, the analyzing project should be a Java Web Application which is compatible with the supported version of the Java language from SonarQube.

The intended users of the Secure codecity Framework are software developers who should have a basic knowledge on software security up to some extent in order to use the tool. Subsequently, the user should analyze the source code of the relevant analyzing software project or project component using SonarQube and identify the security bugs categorized with respect to OWASP T10 [38]. In SonarQube, security bugs are represented as Vulnerabilities. Thereafter, the identified vulnerabilities should be input into the (relevant module) of the Secure codecity Framework. In the case of SonarQube does not encounter any security bugs with respect to OWASP T10, the framework will not be able to show vulnerability severity levels.

SonarQube has the capability of identifying different types of vulnerabilities. The user should configure SonarQube before using the framework.The source code should be completed to a certain level before it is fed to the system since the graphical visualization is based on the current source code.

CodeCity concept is used to visualize the software since it is easy to represents software metrics to the user. Modifications has been done to the codecity concept in order to give clear understanding to the user on security vulnerabilities in each levels of a software project.

3.4 Secure CodeCity Framework Approach

The task of finding security vulnerabilities inside the source code is not much difficult. But compare the severity of vulnerabilities in each class or method is difficult. To do that we have to get vulnerabilities of each and every file and have to calculate the risk separately. It is much time consuming. Although we can find the vulnerable classes or methods, we have no idea about which class or method should we give priority. To address these issues, we introduce Secure CodeCity, 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 overall idea about vulnerabilities, countermeasures and provide quick navigation to IDE to edit the code.

Secure CodeCity organizes source code into a 3D scene in order to take advantage of human spatial memory capabilities and help one better understand. By extending into 3D space in to 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.

Our design goals with Secure CodeCity are threefold. We aimed to create a vulnerability visualization tool that helps user in becoming familiar with vulnerabilities in existing codebase. 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. Saito et al. [55] examined the learning effects between a visual and a text-based environment on teaching programming to beginners. Their results deemed the visual environment as a more suitable option for teaching beginners. Also, Secure CodeCity offers code interaction from an exocentric and an egocentric perspective, combining the benefits of both interaction modalities. The addition of the ego-centric view is the key distinction between the current work and CodeCity [3], as this additional view makes examining the code at different granularities of vulnerabilities possible.

3.5 Secure CodeCity Framework Architecture

3.5.1 Overview of Secure CodeCity

The input to the Secure CodeCity is a standard Java project. As shown in the figure above, the visualization of security vulnerability related information is triggered at three levels of views, namely First Level View,Second Level View and Third Level View. Each descending level view is generated as a result of a selection action which is performed at the previous level, and each descending level view visualizes deeper but more restrictive scoped security vulnerability information related to the input.

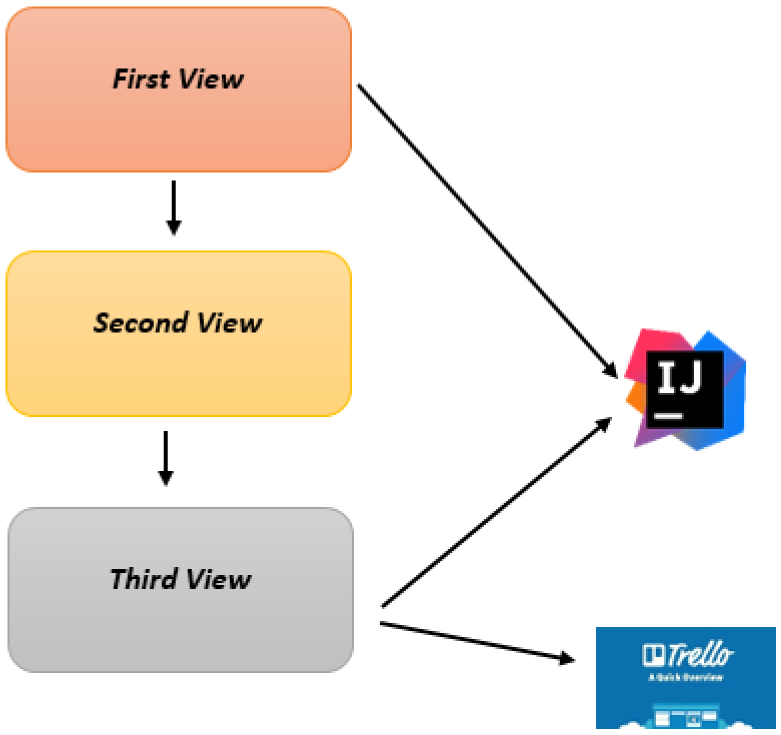


Figure 3.1 : High Level View of Secure CodeCity

3.5.2 First Level View of Secure CodeCity

This is the initial view from which the users can get particular security vulnerability information related to the source code.First Level View visualizes the source code of input project as a 3-dimensional 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 source file, while the height of a building represented according to number of lines of the source file.The colour of a building which represents a class varies according to the overall severity level of vulnerabilities,Security Remediation Effort, Security Rating, Cognitive Complexity and the number of authors in that particular class. Zoom/ Move / Rotate operations could be performed on the 3D city view. Further Total number of Vulnerabilities , Total number of Issues, Security Remediation Effort and Total number of developers of Class could be easily taken by selecting the particular building in First Level Visualization. Also particular selected file could be open in IDE and Source code analyser tool.

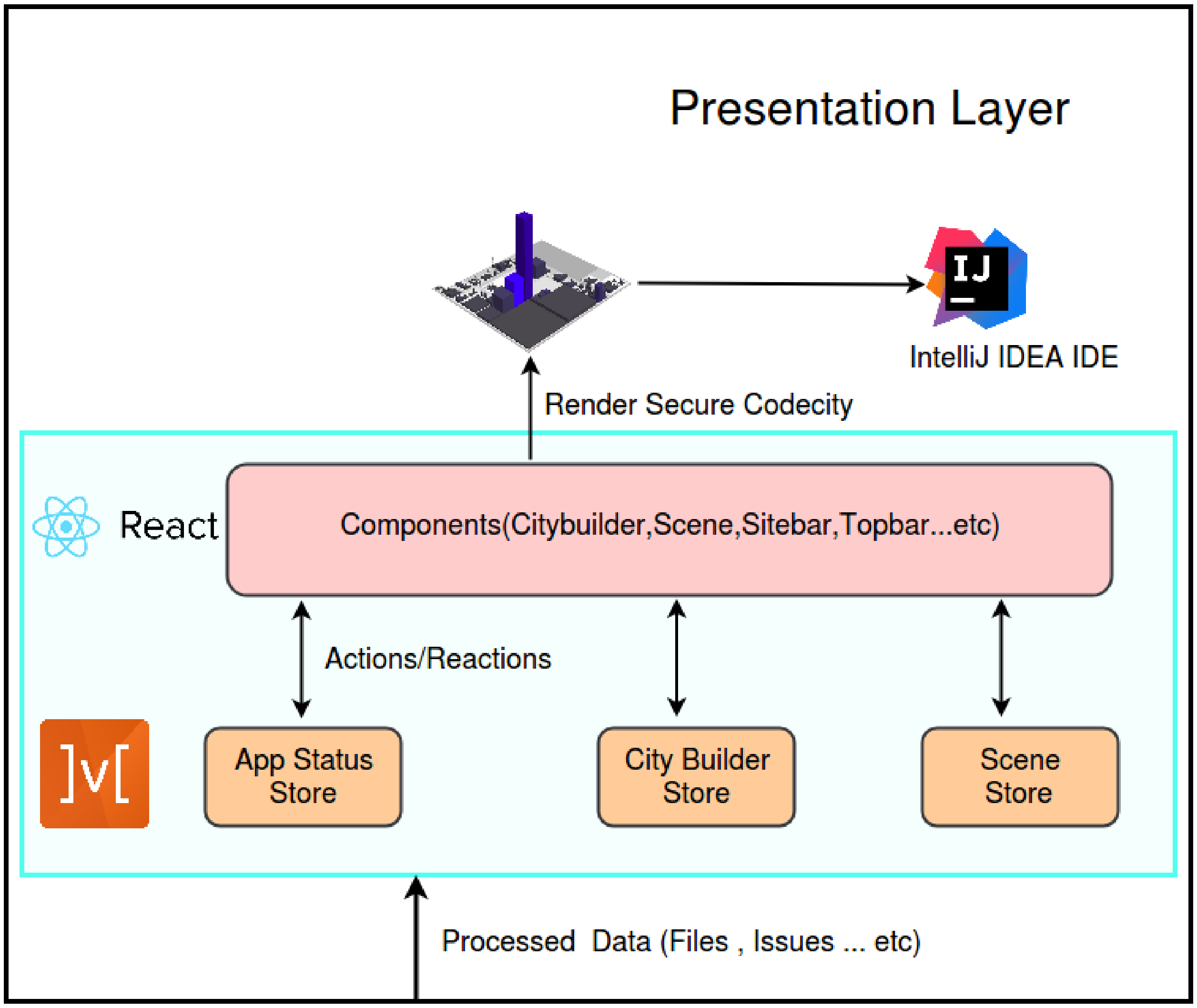


Figure 3.2 : Presentation Layer of First Level View

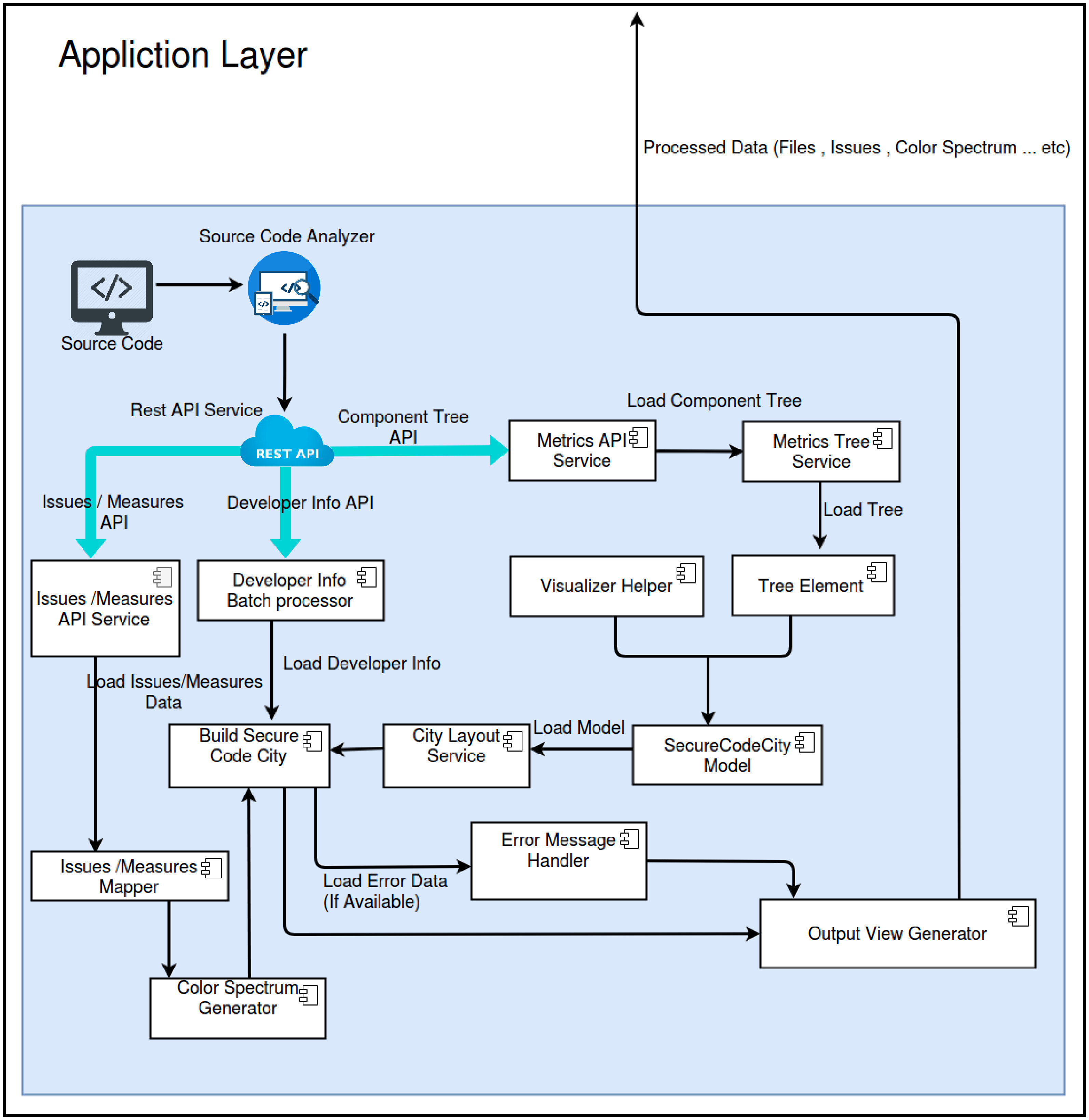


Figure 3.3 : Application Layer of First Level View

3.5.2.1 Presentation Layer - Graphical User Interface

This layer is visualization of source code as a 3 dimensional city where the districts in the city represents packages and buildings represents classes.Informations about Security Vulnerability Severity, Security Remediation Effort, Security Vulnerability Rating, Issue, Cognitive Complexity and number of developer details could be taken using the color spectrum of the buildings and options would be provided in the interface.The information about User Interfaces would be managed using stores.

3.5.2.2 Application Layer

Application Layer of the First Level View is used to get results from source code analyser tool and would map those details to Tree Element Node.Tree Element node is used to store processed data that is taken from source code analyser tool. *"SecureCodeCity Model"* would be created by parsing *"Tree Element"* to *"Visualizer Helper"*. *"Issues/Measures API Service"* Loads the Issues/Measures Data to *"Issues/Measures Mapper"* component.Color of the building would be applied according to severity level of vulnerabilities,Security Remediation Effort, Security Rating, Cognitive Complexity and the number of authors in that particular class that would be generated using *"Color Spectrum Generator"* Component. Eventually Secure CodeCity is built by loading city model and by loading generated color spectrum.Also Error messages would be shown accordingly by using the *"Error Message Handler"* If there are any error occurred while rendering or calling API .

1.Metrics Pre-processor

Static code analysis tools would be used to obtain Component and File details of project to generate Tree Elements representing the project structure. To store details about the extracted details from source code analyser, TreeElement would be used in secure code city model. TreeElement is a node where it contains id, key, measures, name, path, boolean isFile, parent (TreeElement type), array children attributes. The extracted details from source code analyser is mapped to tree elements to create the project structure which would be used to build secure code city first layer view. Here multiple API calls are performed in order to get all Files and Components related details.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. Hence such details are stored in measures attribute (measures is declared as a list) of the TreeElement node. The extracted details from source code analyser would be taken to build secure code city using API service, which would be more suitable because it would ease the implementation. “Component API”, “Component Tree API” and “Component Metric API” data would be processed and would store data in “Tree Element”. Generated tree element would use the “Visualizer Helper” to build “Secure CodeCity Model”.

2. Issues Pre-processor

Source code analyser tools would be used to obtain Security Remediation Effort, Security Rating Details, Security Vulnerability Details of given Source File. Security Issue Details would be Processed according to need of the application , would send those as JSON object. This would be done in “Issue API Service” Component, The result would be mapped to related file in “Issue Mapper” Component.

3. Building and District Generator

The metrics results related to class details and package details would be stored in tree node data structure in metric pre process stage. All building, Packages related metric details would be taken from tree node and would generate building and district using the help of *“Visualizer Helper”* module. *“Visualizer Helper”* component is built as separate node module and would import as package module.In *“Visualizer Helper”* Component CodeCity 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. *“Secure CodeCity Model”* component which built would be created by Parsing *“Tree Element”* to *“Visualizer Helper”* Component. *“City Layout Service”* would load Secure CodeCity model and will build Secure CodeCity by using *“Build Secure CodeCity”* Component.

4. Building Color Visualizer

The color of the building represented according to Security Vulnerabilities Severity, Security Remediation Effort, Security Vulnerability Rating, Cognitive Complexity and Number of author of the Class. The related security measure could be chosen from combo box,the building color would be represented according to severity of the class without reloading the page. The color would spread from Green to Red, Green color building would represented classes which have low severity while Red color building would represented classes which have high security severity. Security Vulnerability Rating of a class calculated as A, B, C, D and E where A would represent 0 Security Vulnerabilities, B would represent at least 1 Minor Vulnerability, C would represent at least 1 Major Vulnerability, D would represent at least 1 Critical Vulnerability and

E would represent at least 1 Blocker Vulnerability. The Rating system and Standard color for those rating would be taken from sonarqube standards. Security Remediation effort would be calculated as sum of the time in minutes to rectify each security vulnerability in class. Security Remediation Effort Could be changed according to vulnerability types. An algorithm would be used to generate color of the building according overall class remediation effort variation. For low Security Remediation Effort green color would be chosen and for high Security Remediation Effort Red color would be chosen.The coloring algorithm generate color spectrum from green to red. For Visualizing Security Vulnerabilities Severity and Cognitive Complexity in First View would also use coloring algorithm accordingly. The *"Color Spectrum Generator"* component would be used implement the coloring algorithm for visualizing color of building according to Security Vulnerabilities Severity, Security Remediation Effort, Security Vulnerability Rating and Cognitive Complexity.

5. File Hierarchy Generator

The selected districts, buildings related packages,files are shown in file hierarchy in the right of the application. The related file would be highlighted in file hierarchy if a building is selected in secure code city and vice versa. The file hierarchy is generated using the values which contains in tree node data structure. This would help user to view data related file hierarchy in two dimensional View. The File Hierarchy Generation would done in *"Build Secure CodeCity"* Component and would parse File Hierarchy to *"Output View Generator"* Component.

6. The Error Message Generator

The related error messages would be shown if there was an error occurred. This would ease the users to use the system. Error Message Generation would done in *"Error Message Handler"* Component.

7. The number of author details batch processor

The number of author related details would be extractracted from source code analysis tool for each file and each result would be processed batch wise to get number of author details for each file.This would be done in *"Developer Info Batch Processor"* Component. The color of the building would be represented according number of author. The green color building would represented classes with high number of authors while the red color building would represented classes with low number of authors. The Color Generation would be done in *"Color Spectrum Generator"* Component.

8. IDE Navigator

The most vulnerable class, vulnerable classes could be chosen by aid of color spectrum and could get the number of vulnerability, Remediation Effort, Security Vulnerability Rating, Cognitive Complexity of that particular class. If the user wants to navigate to that particular file in IDE, User could navigate to that particular file in IDE by clicking “Open IDE” button located in top bar.

9. SonarQube Navigator

The most vulnerable class, vulnerable classes could be chosen by aid of color spectrum and could get the Number of Vulnerability,Security Remediation Effort of that particular class. If the user wants to navigate to sonarQube, user could open file in sonarQube to see issue in 2D View by clicking “Open File” button located in top bar.

3.5.3 Second Level View of Secure CodeCity

When a selection of a building in the first view(a class) is triggered, the second view appears.This view also has buildings,where each building depicts a method in the class related to the selected building.The colour of a method building varies according to the overall severity level of vulnerabilities in that particular method.This view can identify all the methods ,in which a particular OWASP vulnerability category is present, in the selected class.Also this view can share some other useful information such as number of OWASP vulnerabilities in a particular class, number of BLOCKER,CRITICAL,MAJOR,MINOR and INFO vulnerabilities as a percentage and vulnerability distribution among the selected class using different charts.

Second level view depicts as a room view. As we discuss earlier. First level is based on codecity concept and it has building views. In order to come to second level, user has to navigate to inside the buildings. Because methods are inside classes. Normally rooms are available inside buildings. So it's suitable to use second level as a room view. Methods are representing as building blocks on a table. Building structure is used as same as the class representation because it is easy to show software metrics like number of lines. Building blocks related to methods are represented on the table. Three walls of the room is used to show charts which represent different aspects of vulnerabilities. Here we suggest a concept like different walls depicts different aspects of the identified vulnerabilities and user can aware of them in one view.

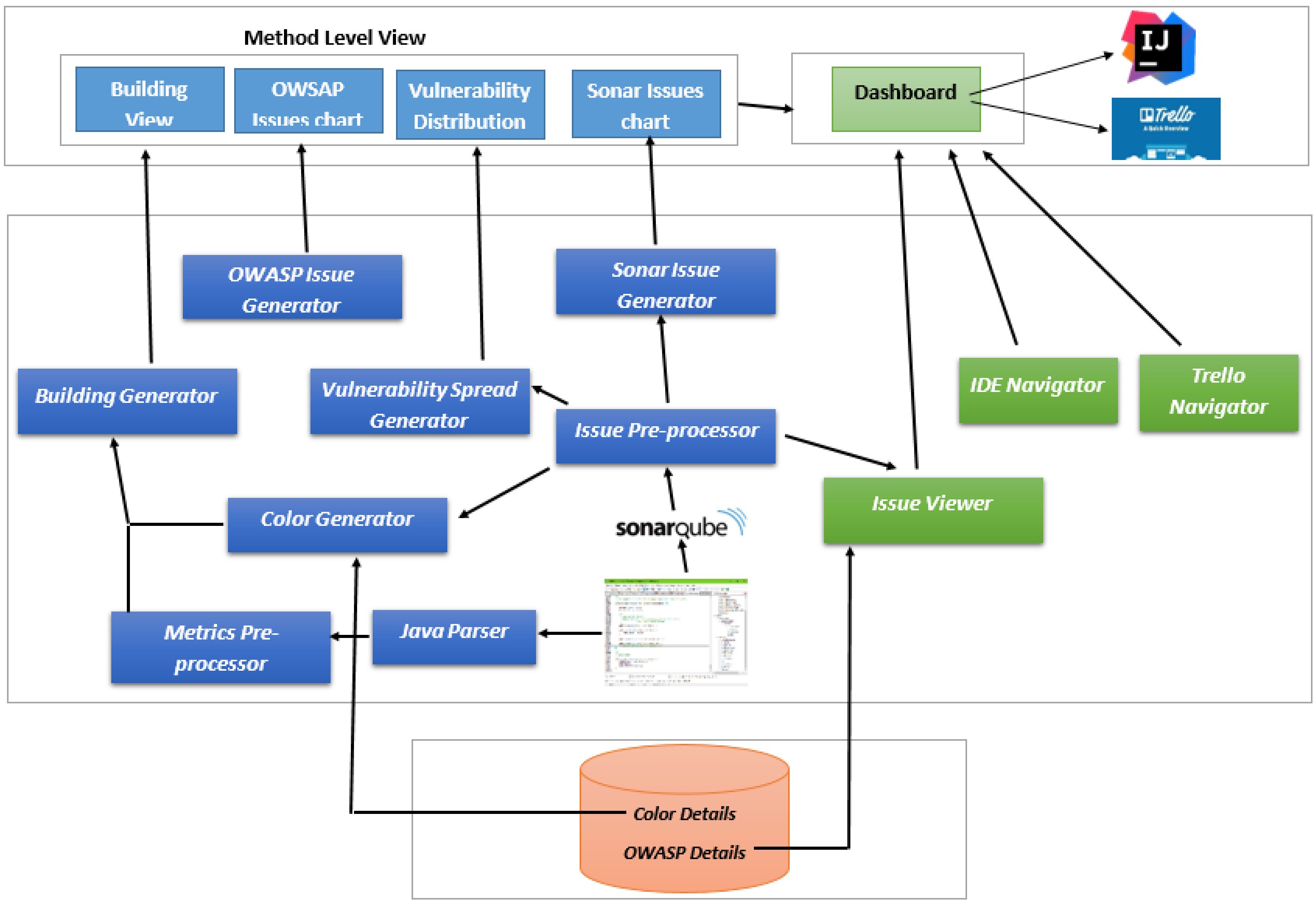

Method Level View
Building View
OWSAP Issues chart
Vulnerability Distribution
Sonar Issues chart
Dashboard
OWASP Issue Generator
Sonar Issue Generator
Building Generator
Vulnerability Spread Generator
Issue Pre-processor
IDE Navigator
Trello Navigator
Color Generator
sonarqube
Issue Viewer
Metrics Pre-processor
Java Parser
Color Details
OWASP Details

Figure 3.4 : Architecture Diagram of Second Level View

3.5.3.1 Presentation Layer

This layer includes the user interfaces which is a 3D room view. Building view is represented on a table inside the room and another three charts are represented on the walls of the room view. So there are four main types like Building view, OWASP Issue Chart, Vulnerability Distribution and Sonar Issue chart.

3.5.3.2 Application Layer

1. Metrics Pre-processor

This module is used to get number of methods and names of methods of the classes of a project. It is used to get the starting line and ending line of particular method. Metrics that were generated by Metrics generator are used as a input to the building generator and vulnerability spread generator.

2. Issues Pre-processor

This module is used to get vulnerability issues 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 3D building view for the methods of selected class. Each building represents a method. The height of the building shows the number of lines of the method. The base size of the building shows the number of vulnerabilities of the method.The metrics for generating buildings were taken from metrics Pre-processor.

4. Sonar Issue Chart Generator

This module is used to represent the severity level of vulnerabilities from sonarQube aspect. Here we used five severity levels which are introduced by SonarQube as MINOR, BLOCKER, CRITICAL, INFO and MAJOR.Pie Chart is generated by this module and it includes how much percentage does the methods in the class takes for each severity level issue.

5. Vulnerability Distribution Chart Generator

This module is used to represent the vulnerability spread of the class. Here we represent the vulnerable line of each method by using Scatter chart. Inputs for this module is taken from Issue Generator and Metrics Pre-processor.

6. Owasp Issues Chart Generator

This module is used to represent the vulnerabilities related to OWASP top 10. It represents how many issues are in the class relevant to each OWASP category. Inputs for this module is taken from Issue Pre-processor.

7. Color Generator

This module is used to assign colors for the buildings according to the risk level. OWASP Risk Rating Methodology is used to calculate the severity level of vulnerabilities in each method. It assigns numerical values for the parameters (Ex: Exploitability, Prevalence..) which are used to rate the issue category. The equation for calculating the severity level is below.

$$\text{Risk} = ((\text{Exploitability} + \text{Prevalence} + \text{Detectability}) / 3) * \text{Technical}$$

If there are vulnerabilities more than one within a method, risk is calculated as below.

$$\text{Risk} = \frac{\text{Get sum of the risk value of each vulnerability}}{\text{Number of vulnerabilities}}$$

Buildings are assigned a color according to the value obtained by equation. If there is a method which has no any vulnerability, it is colored as green color. If there is a method which has vulnerabilities but it does not provide any owasp category related to this, it will be colored as brown color. All the other methods which have vulnerabilities will be colored from light red to dark red according to the risk in descending order.

8. Vulnerability Filter

This module is used to filter vulnerabilities as we want by selecting required types.It will represents building view according to the selected types. Inputs for this module is taken from Issue Generator and Metrics Pre-processor.

3.5.3.3 Data Layer

This will consist of a mapping of countermeasures for Security vulnerabilities categorized to OWASP top 10. And also it is included the color details which are set for severity levels of vulnerabilities.

3.5.4 Third Layer View of Secure CodeCity

As shown in figure 3.4, selection of a method in the method level view will cause navigation from the second level view to third level view.Since the drilling down of source code information is limited to three levels, third level is the ultimate level of displaying information.All the vulnerability details in the selected method, are shown in third view,along with the countermeasures suggested by OWASP. It is possible to navigate to the exact line in the IntelliJ IDEA IDE where a particular vulnerability begins, and to create and post an issue cards on behalf of Trello,which is a collaboration tool.

3.5.4.1 Presentation Layer

This layer includes a dashboard which can display vulnerability details of a selected method.It shows the information such as introduction to the vulnerability, effort to rectify the vulnerability, OWASP category,countermeasures etc. regarding a particular vulnerability.

Third level view provides means to navigate to a particular line in IntelliJ IDEA IDE in the need of rectifying a vulnerability and to create issue cards and post it to Trello. IntelliJ IDEA IDE and Trello are included in the presentation layer because the users of the system can interact with those.

3.5.4.2 Application Layer

1. Issue Viewer

This module is used to get vulnerability details of a particular method in a class, and corresponding OWASP countermeasures for those vulnerabilities . It provides details related to the particular vulnerability like severity,message,debt,effort and so on. These values are extracted from sonarqube.And the OWASP countermeasures are retrieved from data files resting at the data layer

2. IDE Navigator

This module is used to navigate to a particular line number in IDE.When a vulnerability is selected by the user, this module provides means for the user to rectify that vulnerability, by navigating to the line of code in the IDE, which is the starting line of the vulnerability.

3. Trello Navigator

This module is used to create and post issue cards on behalf of Trello. Trello is a collaboration tool that organizes projects into boards. In one glance, Trello can give information related to a particular project, such as what's being worked on, who's working on what, and where something is in a process.Whenever a task/issue is identified, it can be posted to Trello board by creating a card for that issue/task.Trello navigator can create a card for an identified issue and post it to the Trello board.

3.6 Summary

In this chapter, the design of the Secure CodeCity framework has been discussed in detail covering main application modules, components and the functionalities of those application components. First View is used to provide the security vulnerability information for each class of the input project , and user is allowed to drill down to method view to get the idea about method level vulnerabilities and related information.From method view it navigates to the dashboard which contains more details about the identified vulnerabilities, From dashboard user can navigate to the IDE to edit the source code for the rectification of the security issue, or user can report the issue to Trello.

Chapter 4 : Implementation

4.1 Introduction

This chapter explains the development approaches taken in the implementation of the proposed framework described in Chapter 3 with the tools and technologies used for the development. A detailed description of the implementation of each component in the Secure Codecity Framework architecture is described under sub-modules with the issues and challenges occurred and the decisions taken during the development process. The reasons for the selection of technologies and tools used in each component is also described in this section.

4.2 Tools and Technologies

The framework was developed as a standalone application which render in the browser. It consists of a different levels and some levels are used 3D visualization. Threejs library is used with JavaScript to 3D implementations. The First Layer of the application is built using TypeScript, React and MobX and tested using mocha, chai, sinon and enzyme. The rest of the project is built using JavaScript programming language with HTML and CSS.
The first layer of the application has developed using the typeScript, React, and MobX , because the Secure CodeCity should run fast and reliable manner to visualize the security related data. TypeScript is used because it is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer

environment for spotting common errors as you type the code.The First layer of the application was divided to components such as citybuilder, scene, sidebar, topbar etc. and implemented those components using React.To manage the React states, MobX was used. JavaScript Testing Frameworks are used to test the application, for frontend enzyme and for backend chai and mocha. Also, Sass was used to style the front end.webpack was used to bundle JavaScript files for usage in a browser.
The metrics pre processor for method level was built using Java 8 programming language with Spring Boot and Maven technology. The main reason behind selecting JavaScript for the main development of framework is a rich set of libraries which can be used to communicate with 3D techniques and allow to run in browser. The additional reasons for selecting JavaScript programming language is, it is a popular, robust, secure, and high-performance language used in the industry.
The developed software application Secure CodeCity is not a follow-on member of a product family, it depends on outputs from the MS TMT [25] and SonarQube [36].

4.2.1 Static Code Analysis: SonarQube

In order to identify the Security vulnerabilities of a particular software application and to obtain the some of software metrics, the Secure CodeCity Framework user needs to analyze the source code using SonarQube. The vulnerabilities identified as security bugs are categorized with respect to OWASP T10 [13] by this tool. SonarQube is a static code analysis tool as well as a code quality measuring tool which has been widely used in the software security domain. Categorization of software bugs into OWASP T10 is the additional reason for selecting this tool for the proposed approach of the framework as identified in Chapter 2.

4.2.2 Java Parser

In computer technology, a parser is a program, that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts that can then be managed by other programs.[56] In this project, a java parser, which is a set of tools and libraries to parse a given Java source code into low level components, is used. Java parser has the ability to derive the abstract syntax tree(AST) related to a Java source code, as well as to break down Java source code to methods, statements with particular keywords etc.

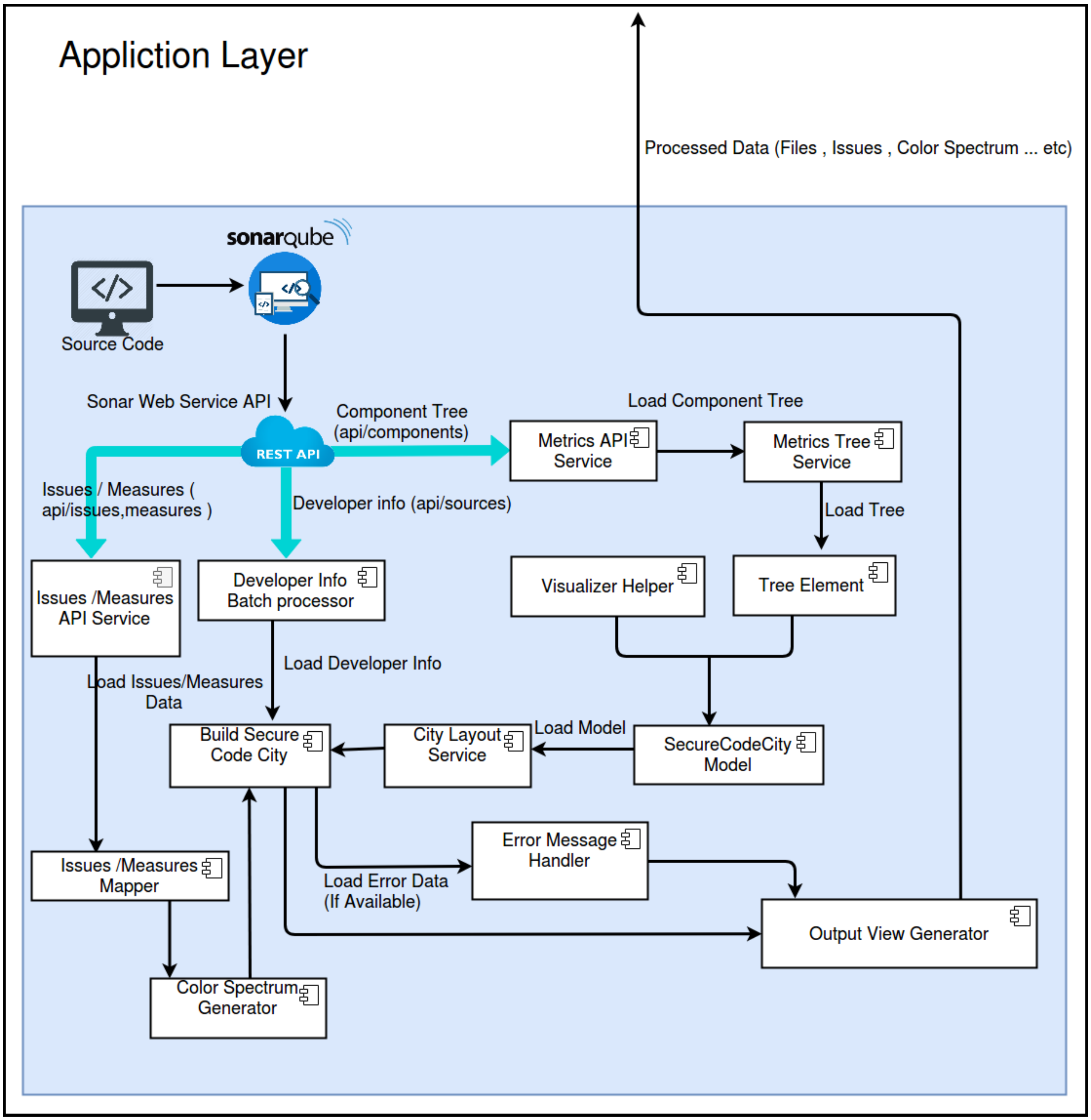


Figure 4.1 : Application and Data Layer of First Level View

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 where Security Vulnerability Severity, Security Remediation Effort, Security Vulnerability Rating, Issue, Cognitive Complexity and number of developer details could be taken using the color spectrum of the building. Also specific source file details such cyclomatic complexity, number of line of code, Security Remediation Effort, Security Vulnerability Severity, Security Vulnerability Rating, Cognitive Complexity and number of developer could be taken by selecting the building. In order to make these process efficient, the application is built as Single Page Application using React and MobX. The React Components and functionalities of first layer were tested using enzyme, chai, sinon and mocha.

1.Metrics Pre-processor

SonarQube provides API to get metric details related to the scanned project, where Component and File details could be taken to generate Tree Elements representing the project structure. To store details about the extracted details from sonarqube, TreeElement is used which is implemented using the TypeScript Classes.TypeScript is used to implement project because it supports OOP concepts hence classes, inheritance features could be used to build the application. TreeElement is a node where it contains id, key, measures, name, path, boolean isFile, parent (TreeElement type), array children attributes. The extracted details from sonarQube is mapped to tree elements to create the project structure which would be used to build secure code city first layer view. Here multiple API calls are performed in order to get all Files and Components related details.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. Hence such details are stored in measures attribute (measures is declared as a list) of the TreeElement node. "*Component API"*, "*Component Tree API"* and "*Component Metric API"* data would be processed and would store data in "*Tree Element"*. Generated tree element would use the *"Visualizer Helper"* to build secure code city model.The application is built as Single Page Application using the React and MobX, so there we using states to store values and props are used to communicates among components.

2. Issues Pre-processor

SonarQube provides facility to extract vulnerability details, Security Remediation Effort, Security Rating Details of any given File via Sonar API. It sends output details as a JSON object. Each json object is processed and security related details are stored in tree node.This would be done in *"Issue API Service"* Component, The result would be mapped to related files i.e. related tree nodes which contain in *"Build Secure CodeCity"* by use of *"Issue Mapper"* Component. For API calls node JS is used and for testing the those API chai, mocha and sinon are used. The details are processed according to various needs of the project using TypeScript.

3. Building and District Generator

The metrics results related to class details and package details would be stored in tree node data structure in metric pre process stage. All building, Packages related metric details would be taken from tree node using typeScript and would generate building and district using the help of *"Visualizer Helper"* module. In *"Visualizer Helper"* Component CodeCity 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. *"Secure CodeCity Model"* component would be created by Parsing *"Tree Element"* to *"Visualizer Helper"* Component. *"City Layout Service"* would load Secure CodeCity model and will build Secure CodeCity by using *"Build Secure CodeCity"* Component.The Functionalities of those component were implemented using typeScript and node JS. And Wrote unit tests for those using chai, mocha and sinon.

4. Building Color Visualizer

The color of the building represented according to Security Vulnerabilities Severity, Security Remediation Effort, Security Vulnerability Rating, Cognitive Complexity and Number of author of the Class. The related security measure could be chosen from combo box,the building color would be represented according to severity of the class without reloading the page. The color would spread from Green to Red, Green color building would represented classes which have low severity while Red color building would represented classes which have high security severity. The *"Color Spectrum Generator"* component which is implemented using typeScript, would be used implement the coloring algorithm for visualizing color of building according to Security Vulnerabilities Severity, Security Remediation Effort, Security Vulnerability Rating and Cognitive Complexity. The related data would parse by using store of the MobX among the components.The Frontend built using React, was tested using enzyme.

5. File Hierarchy Generator

The selected districts, buildings related packages,files are shown in file hierarchy in the right of the application. The related file would be highlighted in file hierarchy if a building is selected in Secure CodeCity scene and vice versa. The file hierarchy is generated using the values which contains in tree node data structure. The File Hierarchy Generation would done in *"Build Secure*

CodeCity" Component which is implemented typeScript, and would parse File Hierarchy to *"Output View Generator"* Component. The Frontend built using React was tested using enzyme.

6. The Error Message Generator

The related error messages would be shown if there was an error occurred. For instance, if browser is not support for WebGL then it would popup an error messages stating the webGL issue. If sonar API are not available to obtain details then it would popup an error message stating sonarQube API issue. Error Message Generation would done in *"Error Message Handler"* Component which would take "error" messages from *"Built Secure CodeCity"* Component. Those Component were implemented using typeScript and React.

7. The number of author details batch processor

The number of author related details would be extractracted from sonar API for each file and each API JSON response would be processed batch wise to get number of author details for each file. This would be done in *"Developer Info Batch Processor"* which is implemented using typeScript and nodeJS, and tested those implementation using chai, mocha and sinon. The color of the building would be represented according number of author. The green color building would be represented classes with high number of authors while the red color building would be represented classes with low number of authors. The Color Generation would be done in *"Color Spectrum Generator"* Component which is implemented using typeScript.

8. IDE Navigator

The most vulnerable class could be chosen by aid of color spectrum and could get the number of vulnerability, Remediation Effort, Security Vulnerability Rating, Cognitive Complexity of that particular class. If the user wants to navigate to that particular file in IDE, User could navigate to that particular file in IDE by clicking "Open IDE" button located in top bar. The path to the source file would be taken from the scene props.

9. SonarQube Navigator

The most vulnerable class could be chosen by aid of color spectrum and could get the Number of Vulnerability,Security Remediation Effort of that particular class. If the user wants to navigate to sonarQube, user could open file in sonarQube to see issue in 2D View by clicking "Open File" button located in top bar. The path to the source file of selected building would be taken from the scene props.

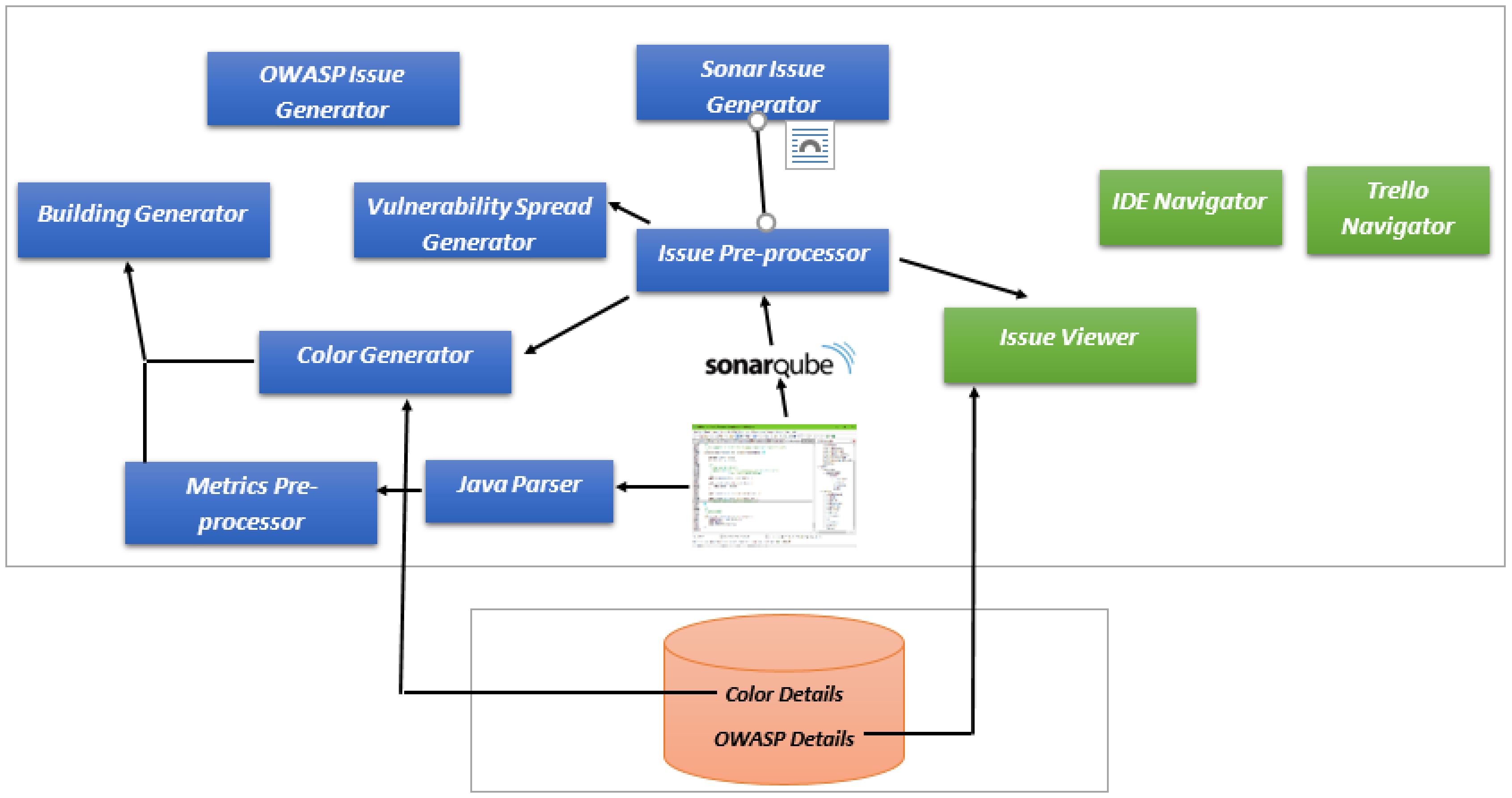

OWASP Issue Generator
Sonar Issue Generator
Building Generator
Vulnerability Spread Generator
Issue Pre-processor
IDE Navigator
Trello Navigator
Color Generator
sonarqube
Issue Viewer
Metrics Pre-processor
Java Parser
Color Details
OWASP Details

Figure 4.2 : Application and Data Layer of Second Level View

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 methods details, source code will be fed to java parser and extract method name, size and method lines details. It is processed using Java with visitor design pattern.
Rest API service is created using Spring Boot to send those processed metrics values as a JSON format to building generator component.

2. Issues Pre-processor

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

3. Building Generator

The metrics results related to method details which are coming as a JSON object from Metrics Pre-processor will be processed to extract the required information to build view using JavaScript.Some vulnerability details are also needed to build this view and those details are extracted from Issues Pre-processor. View is generated using Three.js library.

4. Sonar Issue Chart Generator

The details taken from Issues Pre-processor and metrics pre-processor are used to create pie chart which appear on the wall of the room view.Canvas-js charts library and JavaScript are used to generate chart.

5. Vulnerability Distribution Chart Generator

To generate this chart which appear on the wall of the room view , metrics pre-processor details and issue pre processor details are used. Canvas-js charts library and JavaScript are used to generate chart.

6. Owasp Issues Chart Generator

The details taken from Issues Pre-processor are used to create bar chart which appear on the wall of the room view.Canvas-js charts library and JavaScript are used to generate chart.

7. Color Generator

Color details will be stored in data layer. This module calculate the risk value and get the related color code from color details file. JavaScript is used to implement these functionalities.

4.2.5 Third level

1. Issue Viewer

Vulnerability details of a particular method are retrieved through the session.These details are stored in the session when the user is at the second level view. The retrieved details are processed according to various needs of the project using JavaScript.OWASP countermeasures for those vulnerabilities are stored in data layer. This module uses these countermeasures stored in a data file.JavaScript is used to implement these functionalities.

2. IDE Navigator

Whenever a user selects a vulnerability and wants to navigate to the starting line of that vulnerability in the IDE, this component paves the way. The starting line number of the selected vulnerability and path to the selected java file in the first level, are passed as inputs to an API which runs locally. This API calls a bash script function which takes the input parameters as arguments and locate specific line number n the mentioned file path, in Intellij IDEA IDE.

3. Trello Navigator

Trello REST API has capabilities to post issue cards to Trello boards. Whenever a user is in need of reporting a particular security vulnerability as an issue to Trello, Trello Navigator component creates an issue card dynamically by appending vulnerability information together using javascript and post it to Trello using Trello API.

4.3 Summary

This chapter presented the implementation procedure of the entire system with the tools and technologies used in each submodule. The implementation of the design modules in Secure CodeCity Framework architecture explained in Chapter 3 were technically presented as an integration of them with the reasons behind in selecting relevant tools and technologies.

Chapter 5 : Testing and Evaluation

5.1 Testing

The testing procedure was conducted as a strategy to ensure that the secure codeCity product operates as intended in the specification. It can be realized under two main categories namely,functional testing and non-functional testing. Functional testing 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 was performed using the chai, mocha, sinon and enzyme.The First Layer View of the application was built using React, MobX and node JS, which was tested using Enzyme and backend developed using node JS tested by chai, mocha and sinon where Asynchronous API calls were also being tested using those javaScript testing frameworks. The manual testing also carried out for 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 analyzing benchmark projects with SonarQube itself and the whole system and check weather system generates the expected outputs to verify that all components together works 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 SecurityShepherd and WebGoat used for the evaluation purpose in the submodule 5.3 was selected for this purpose.

5.2 Evaluation

This research intends to present a novel mechanism and a proof-of-concept framework for visualizing security vulnerability information of java projects First, the evaluation is carried out to measure the overall correctness of Secure CodeCity.Second, it was evaluated whether the results produced by Secure CodeCity is time efficient when compared with SonarQube which is a static code analysis tool. Finally, the usability of Secure CodeCity is compared with the same tool. Three hypotheses (see Table 5.1) have been formulated to assess the above claims, and in that way, we plan to answer RQ2: When software practitioners use such a framework to perform static code analysis tasks, can we witness an improvement over the classic static code analysis tools such as SonarQube, in terms of accuracy, time efficiency, and usability.

5.2.1 User Evaluation Experiment

The user evaluation experiment has been conducted based on the Between Subjects Design, which is one of the well-known experimental design approaches in Software Engineering [30]. The purpose of user evaluation experiment was to carry out a multi-fold evaluation of Secure CodeCity in terms of correctness when performing the tasks, the time to complete tasks and the overall usability of Secure CodeCity when compared with SonarQube.The research population of this experiment consists of 22 subjects where some of them are UCSC postgraduates in computer science who have been working in the software industry for a time period of 1-4 years , while some of them are industry experts who have been working in the software industry for more than 4 years.The inclusion criteria were mainly based on working experience in the software industry together with the familiarity with static code analysis tool,SonarQube.

From the 22 participants,11 were assigned to the experimental group, and the rest 11 were assigned to the control group. A purely random approach has used to assign the subjects to both experimental and control groups. The experimental group was provided with Secure CodeCity, whereas the control group was supposed to carry out the tasks with the SonarQube tool. Figure 5.1 depicts the industry experience and the familiarity with SonarQube tool in both experimental and control groups. Mean values for the industry experience and the experience with the tools of the both experimental and the control groups are statistically not significantly different.

Null Hypotheses	Alternative Hypotheses

H1o The total correctness score for all tasks is the same across the experimental and control group.	H1 The total correctness score for all tasks is different across the experimental and control group.
H2o The System Usability Scores is the same across the experimental and control group.	H2 The System Usability Scores is different from the experimental and control group.
H3o The time spent on solving all tasks is the same across the experimental and control group.	H3 The time spent on solving all Tasks are different across the experimental and control group.

Table 5.1 : Hypotheses

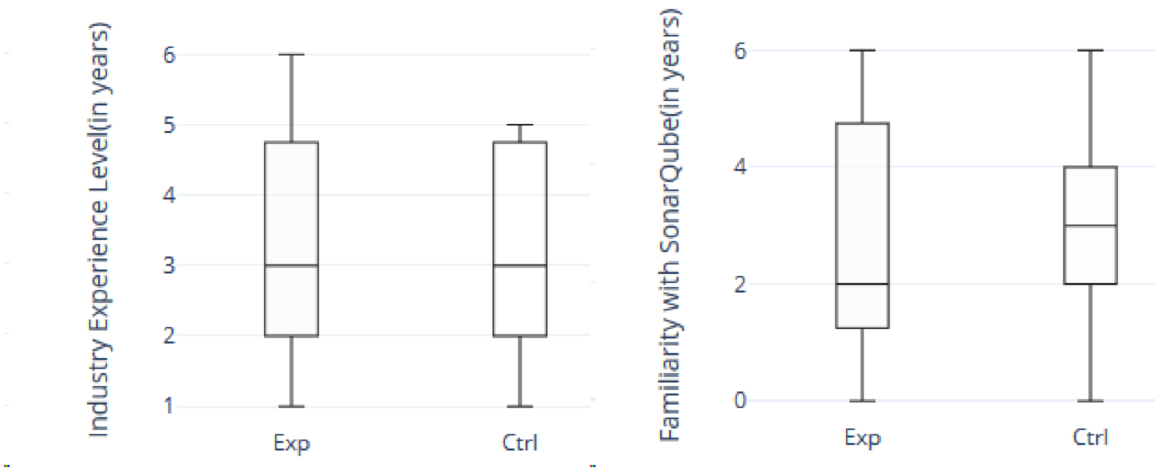


Figure 5.1 : Industry Experience (left) and Familiarity with the Tools (right) for Experimental and Control Groups

User evaluation experiment started with a brief introduction to outline the objectives of the experiment. Then we demonstrated how Secure CodeCity works by selecting a code analysis task (outside the eleven tasks chosen for the user evaluation experiment) and the same task has demonstrated by using SonarQube. By doing that, we were able to communicate the main objective of the user evaluation experiment to the participants in both experimental and control groups.The task list has distributed in the form of a questionnaire, where participants were expected to write down the answer upon completion of each task. Knowing the nature of some tasks, the duration to solve a task has limited to 10 minutes.

To evaluate the usability of Secure CodeCity, it was decided to use Computer System Usability Questionnaire(CSUQ), which was based on Lewis J.R.(1995) IBM Computer Usability satisfaction Questionnaire, with some alterations . CSUQ perfectly fits our requirements as it is technology independent and has tested on different types hardware, consumer software, and websites. Therefore, in addition to performing the tasks, we asked our subjects to record their immediate response. CSUQ is based on a sixteen-item questionnaire, and it was instructed to answer to each item rather than thinking about them for a long time.

The experimental group was instructed to provide answers to the questions as per the experience with Secure CodeCity while performing the tasks. The control group was instructed to answer the questionnaire based on the experience with the SonarQube that they used to solve the same tasks. Then a measurement of the usability has been calculated for each participant for Secure CodeCity and the SonarQube separately.

5.2.2 Selection of Questions

The question list and the scenarios presented in the literature is quite comprehensive and highlights several critical aspects in the field of secure software engineering. Therefore, in this research, we used such questions and scenarios as a frame of reference. Our query interface is capable of overcoming the limitations of the existing approaches in addressing such scenarios.Scenarios are directly linked with quality characteristics and finally linked to goals that are described under GQM. The goals are reached by answering the questions. Thus,we have evaluated the capabilities of our mashup framework to assess the quality characteristics by answering such questions.

We have selected three scenarios from the literature in the context of software evolution. Then questions sets were devised for each of the three scenarios to comprehend them better. The selection of scenarios was based on several levels of the proposed framework. First, the selection was done in a way that it is regarded to the class level(first level) vulnerability information and other important software metrics. For example, questions such as ”Which one is/are the most vulnerable class(*) of the input project, according to the system?” is useful information for a software engineers,tech leads,project manager and etc. Second, some of these questions are straightforward and can answer easily. Q2, Q4, and Q5 are classic examples of that nature. However, more complex questions such as Q7 requires combining several software artifacts to obtain the answers. Finally, we were careful enough not to select questions that are biased to Secure CodeCity by penalizing SonarQube. For example, questions such as ”What is the line of code(loc) value of a given class?” can be easily answered with Sonarqube. However, answering a question like ”What is the most critical method in a given class?” is much more difficult to answer with SonarQube, hence showcase the richness of Secure CodeCity.

5.2.3 Selection of Experiment Subjects

The rationale behind the selection of experiment subjects is that the experiment subjects should be able to perform the tasks based on two open-source Apache projects available in Github in order to strengthen the external validity of our user evaluation experiment . Apache projects have been used in many case studies by several researchers over the last decade or so. One OWASP project which was selected, is OWASP Security Shepherd project, which is available as a OWASP project as well. OWASP Security Shepherd project is a web and mobile application security training platform. Security Shepherd has been designed to foster and improve security awareness among a varied skill-set demographic. The aim of this project is to take AppSec novices or experienced engineers and sharpen their penetration testing skill set to security expert status. It enables users to learn or to improve upon existing manual penetration testing skills.The second project which was selected is WebGoat.WebGoat is a deliberately insecure web application maintained by OWASP,which was designed to teach web application security lessons.The selection of projects was based on the availability of such projects in the public instances of Sonarqube and Jenkins, which are able to perform all the tasks. On the other hand, we ensured that the selected projects are of a reasonable size due to the time limitation in conducting experiments. General description of the selected projects and their sizes are described in the next section.

Scenario 1 : To get the class level vulnerabilities and details of important metrics

More than a few studies confirmed that security vulnerability identification is helpful to steer the software development lifecycle. And it can be useful for the programmers and non-programmers, when these security vulnerabilities can be aggregated together to show the overall picture of the severity levels of each class, and ultimately as a united whole project.

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, in a listview.There can be moments where the details of security vulnerabilities of a particular class can be useful for a programmer.

Therefore, obtaining answers to questions Q1, Q2, Q3, Q4 and Q5 would provide insights related to software vulnerability for software practitioners, in class level.

Q1.Which one is/are the most vulnerable class(*) of the input project, according to the system?
Q2.What is the cyclomatic complexity of class MultilLevelLogin2.java?
Q3.What is the Security Remediation Effort in class MultilLevelLogin2.java?
Q4.What is the line of code(loc) value of class MultilLevelLogin2.java?
Q5.How many vulnerabilities are there in class MultilLevelLogin2.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 others.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 in to sonarQube local server, the vulnerabilities related to the particular class(In this case, MultilLevelLogin2.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 contrast, Secure CodeCity generates the answer for the above question on-the-fly, by tracking all the vulnerabilities in the class MultilLevelLogin2.java, and adding up the remediation efforts of each vulnerability in order to derive the remediation effort of the class MultilLevelLogin2.java, behalf of the user. However, our approach does not aim at developing algorithms which can independently calculate remediation effort of a given class,rather it extracts the results produced by other tools to answer these basic questions.

In answering the questions in Scenario 1, Secure CodeCity also fetches data from SonarQube.Since this reason, it can be argued that the experimental results obtained from both experimental group and control group are not significantly different in terms of correctness and the time.However Secure CodeCity performs more tasks behind the scene in which user 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 experimental group and control group are significantly different in terms of correctness and the time.

Scenario 2 : To get 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 listview. Vulnerabilities of each methods are not shown separately.SonarQube does not provide a direct way to get various security related aspects details like getting owasp related issues, getting MINOR,MAJOR,BLOCKER,INFO and CRITICAL issue percentages and 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 doPost in class Register.java

Experiments were conducted on both of the aforementioned projects.Answering Q6 and Q7 without Secure CodeCity is cumbersome, requiring accessing each method in class and get number of issues with 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 class.

Scenario 3 : To retrieve information related to a particular vulnerability and other accompanied tasks such as Trello Reporting and IDE navigation.

Typical static code analysis tools have the capability of providing a wide range of information related to a particular vulnerability.For delivering more efficient and effective rectification mechanism regarding a vulnerability, more detailed countermeasures to rectify a vulnerability can be provided along with the IDE Navigation and Trello Reporting functionalities.
To effectively analyze this scenario, three key questions have been identified (Q9 to Q11).

Q09.Does system suggest countermeasures to rectify the first vulnerability in doPost method in class MultilLevelLogin2.java ?
Q10.Move to the “Resolve” section of the above mentioned vulnerability(question 9).Can you easily navigate to the IDE to edit the code, in order to rectify the vulnerability?
Q11.Can you report the above mentioned vulnerability(question 9) as an issue to Trello?

To perform a better security vulnerability rectification, countermeasures suggested by OWASP can be used.The questions like Q09 elaborate on this aspect.Enabling the user to instantly locate the starting line of a particular security vulnerability is helpful when it comes to an attempt of security vulnerability remediation.This is addressed via the questions such as Q10.
Another noteworthy advantage of 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, various types of vulnerabilities can be resolved in less time.

5.3 Data Collection and Analysis

In this section, we overview the data collected to evaluate our three hypotheses and the empirical results obtained through the statistical analysis.

5.3.1 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 22 points could obtain if all of them were answered correctly.Similarly, 0 marks allocated for the wrong answers and timeouts.

Secure CodeCity's mean correctness score is 10.73 (standard deviation of 0.467) compared with the mean correctness score of 8.636 in control group (standard deviation of 2.693), and these values were calculated according to the tables D(2) & D(4) in appendix D. Box plot is shown in Figure 5.2 for the overall correctness for the both experimental and control groups. As per the box plot, the 50th percentile of the experimental group is above the 75th percentile of the control group, denoting considerable overall correctness of Secure CodeCity over SonarQube.

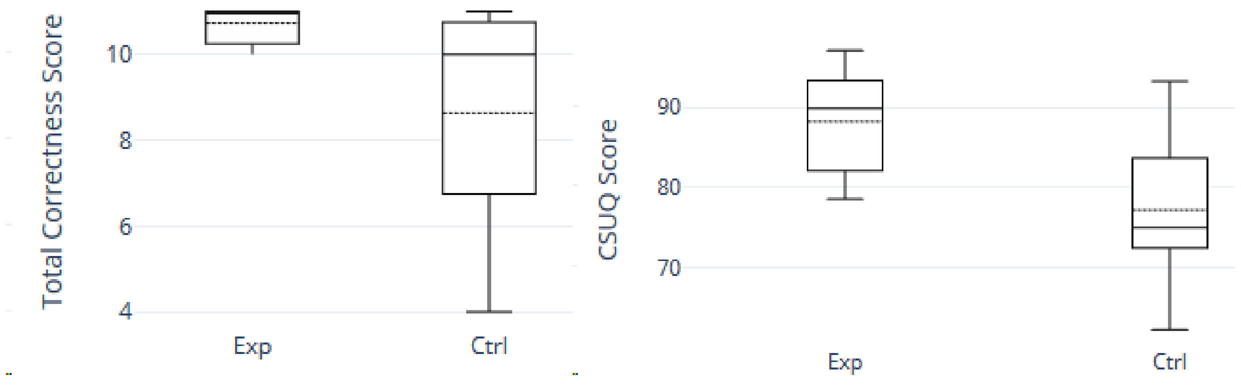


Figure 5.2: Total Correctness Scores (left) and Total Usability Scores (right) for Experimental and Control Groups

5.3.2 Overall usability score of the tasks

We utilized CSUQ[31] and did some improvements to that in order to match for our framework usability evaluation. It comes with 23 questions to evaluate the usability of Secure CodeCity compared with the SonarQube in performing the selected eleven tasks as described in Section 5.3. As mentioned above, the subjects were requested to give a score from 1 to 7 for each of the twenty three questions, based on their degree of satisfaction.The process of getting traditional CSUQ score involves subtracting 1 from the mean of the 23 individual CSUQ items and multiplying that by 100/6 to stretch it out to a 0-100 point scale.Finally, the total score is subtracted from 100 to obtain the final CSUQ score.Though the final CSUQ score is not a percentage value, it gives the score out of 100 and hence, it is an unambiguous way to compare the results.

Secure CodeCity'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), and these values were calculated according to the tables D(5) & D(6) at the appendix D.

Given that the 50th percentile of the experimental group is well above the 75th percentile of the control group according to the box plots in Figure 6, it evidently showed the acceptance of Secure CodeCity over baseline tool, SonarQube,which was used for the evaluation.

5.3.3 Overall completion time of the tasks

Knowing the nature of some questions, we specifically mentioned experiment subjects to not to spend more than 10 minutes on a single task. 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 Q2 and Q4, in particular. However, Secure CodeCity significantly surpasses the SonarQube in answering Q1,Q3,Q6, Q9,Q10 and Q11.

Table 5.2:Average Completion Time for Individual Questions (in secs)

Task/Question	Experimental Group	Control Group
Q1	10.18	441.36
Q2	82	91.18
Q3	14.73	496.82
Q4	85.55	70.45
Q5	28.27	165.18
Q6	12.73	324.64
Q7	15.18	34.82
Q8	15.18	53.45
Q9	16	129.55
Q10	12.91	142.45
Q11	5.91	226.73

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

Secure CodeCity’s mean overall completion time is 297.73s , and it is 1907.09s lower than the that of control group mean score of 2204.82s.And these values were calculated according to the tables D(7) & D(8) in the appendix D.The 75th percentile of the experimental group is much lower than the 25th percentile of the control group according to the box plots in Figure Figure 5.3(a), and that fact clearly indicates the advantage of Secure CodeCity over SonarQube, regarding the efficiency.

The acceptance of the alternative hypothesis confirms that the distribution of the total completion times among the two groups is different. Research question (RQ2) can be answered by accepting hypothesis H1, H2, and H3. It was observed that Secure CodeCity outperforms SonarQube regarding accuracy, time efficiency, and usability.

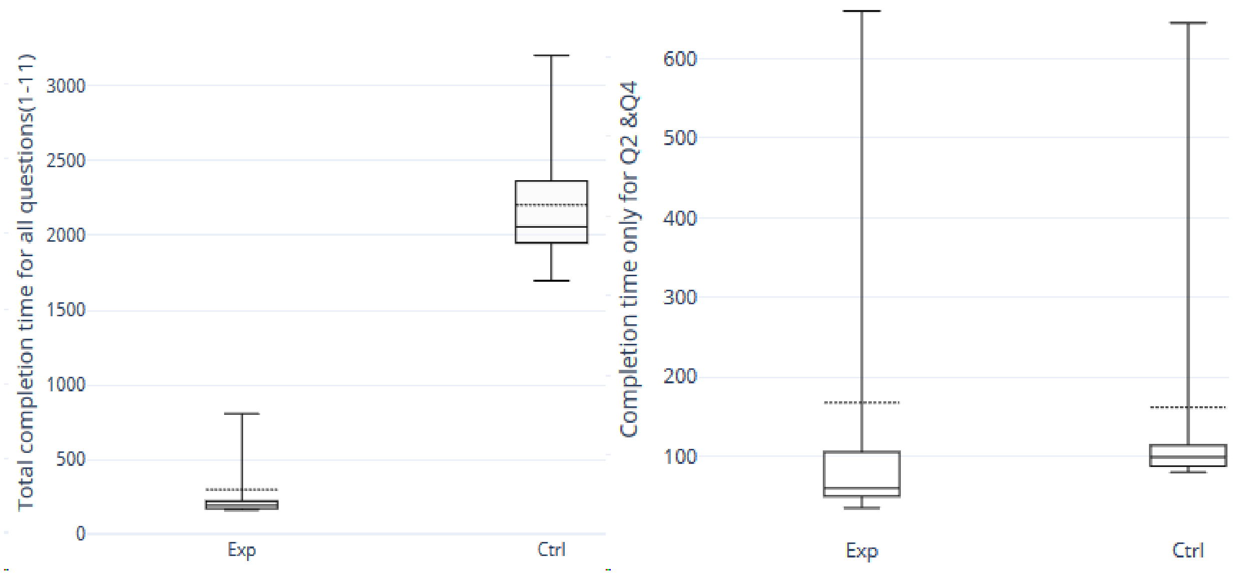


Figure 5.3 : Overall Completion Time for All Questions (a) and Total Completion Time for Q2 and Q4 (b) for Experimental and Control Groups

5.4 On the Threats to Validity

External validity: Threats to external validity are concerned with the generalizability of the results. First, the selected scenarios and tasks in our user evaluation experiment would favor Secure CodeCity, and therefore the results might not be generalized. However, we have minimized the threat by selecting scenarios from the literature, which have been proven important. Second, the selection of experiment subjects may not be representing software practitioners in the industry. As per the inclusion criteria of the subject selection in our research, it was mainly based on working experience in the software industry together with the familiarity of SonarQube, even though some subjects were UCSC postgraduates who have 1-2 years experience. However, we have not specifically looked for the experience in a particular programming language as long as the subjects are familiar with SonarQube.

Internal validity: Uneven distribution of experience level across experimental and control group might bias the results of the user evaluation experiment. However, to minimize the risk, the assignment to the groups have been done in a purely random manner. Both groups have used the stopwatch application on their mobile phones to record the completion time of each task. Therefore, there is a possibility of recording incorrect times by the participants. This can be considered as a minor threat to validity. Therefore, knowing the nature of experiment objects, the time to perform the tasks have been limited to a maximum of five minutes. This issue must be resolved by repeating the same tasks for two different experiment objects selected based on the size of the project. Conclusion validity Being the total number of participants used in the experiment is relatively low, it might influence the results and hence, a threat to conclusion validity.

5.5 Discussion and synthesis

Empirical results of the user evaluation experiment evidently demonstrate that the experimental group was managed to solve the tasks with a higher correctness level in a lower time duration by using Secure CodeCity. One of the key benefits of this approach is that it is capable of dealing with multiple data repositories and multiple underlying algorithms to answer complex questions. A significant difference regarding accuracy and the time efficiency is observed particularly for analysis questions such as Q1,Q3,Q6, Q9,Q10 and Q11 when compared with SonarQube.

However, we do not claim that Secure CodeCity can fully replace SonarQube in every analysis task.The main reason is, Secure CodeCity is depending on SonarQube for data purposes.
For some tasks, Secure CodeCity and SonarQube are showing almost same results in accuracy and time efficiency aspects. For example, finding answers for Q2 and Q4 with and without Secure CodeCity has not shown a significant difference regarding accuracy and the time efficiency.

Figure 5.3(b)shows the time difference between Experiment (mean = 167.5455 and standard deviation = 235.2) and Control (mean = 161.6364 and standard deviation = 168.443) groups when answering Q2 and Q4. Besides, this approach has several noteworthy benefits compared to the SonarQube in the same direction.

Multiple stakeholder support: With the evolution of software systems, different stakeholders such as software developers, testers, and project managers continuously seek various information to perform their daily activities. Most of the existing research mainly focused only on facilitating software developers with their information needs. The main reason is that with the technical and programming knowledge the developers have, they can effectively utilize any framework or tool to get the information they need. In contrast, the programming-free interface of Secure CodeCity allows any nontechnical stakeholder to find various information about the software system. For example, software developers can find information about code quality, whereas project managers can get an initial and summarized idea about the severity of vulnerabilities which can be found in the source code.Therefore, Secure CodeCity is capable of fulfilling the information needs of multiple stakeholders such as developers, testers, project managers and so on. Further, it would enhance and speed up the work of software practitioners by giving them access to a significant amount of information without having to install several tools and to cope with many output formats.

Programming-free composition: Any stakeholder having a conceptual knowledge in the software analysis domain can use Secure CodeCity to answer some basic questions, even without having prior programming knowledge or security related knowledge. Just by seeing the color spectrum and the spread of the colors 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 CodeCity framework.

Chapter 6 : Conclusion

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

6.1 Secure CodeCity Framework 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 SSDLC, as the research problem. To achieve this, initially the background literature was studied to select a suitable software visualization model.Afterward,“CodeCity” was selected as the best software visualization model which can be well aligned with our problem approach.Then an implementation and configuration process was carried out to to develop “Secure CodeCity”,which is the suggested framework for coping up with the aforementioned research problem. Finally, the output was benchmarked with SonarQube,which is a popular static code analysis tool, in order to verify the correctness, efficiency and usability aspects.

The “Secure CodeCity” framework was developed for finding security vulnerabilities occur in the source code and for providing vulnerability details to the user in an attractive manner with navigation and reporting capabilities to the IDE and Trello.The framework mainly focus on software security vulnerability visualization. This approach helps to maintain secure software development methodology. In spite of finding the vulnerabilities of the source code, the framework provides countermeasures for the identified vulnerabilities and prevention techniques for them.This framework can be used while developing the software where the security vulnerabilities and software metrics are identified by providing current source code to the framework.
Implementation of a software application sometimes includes the use of legacy software components. Thus, the Secure CodeCity framework can be used to ensure the security of the legacy components which are lack in software security.

In a security-focused agile development environment, the framework can be used to ensure the security of the working software in each product increment. The particular area of study proposed in this dissertation is exposed to a broad research space. The framework can be used as an aid for researchers in the security domain by enhancing and updating the visualization in consideration with different security aspects.

Considering the fact that if the project can be evaluated as to find possible Security Vulnerabilities occurred in the implementation phase, this framework will be a guide to software developers to follow a Secure Software Development Lifecycle. Also Tech Lead/ Architect could view spread of security vulnerability within project using secure Codecity.

6.2 Future Work

First view works for projects which are developed using many languages which compatible with SonarQube. But second view is only available for java projects. This approach can be further enhanced to enable the second view for any project with any language.

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

And also this can be integrated with agile project management systems like JIRA with more functionalities so that it will be easier for developers to manage issues which are found by using this system.Currently this framework supports Trello, which is also a project collaboration tool.

And furthermore, this framework can be enhanced to be integrated with CI/CD tools such as Jenkins, TeamCity, Bamboo etc.

References

[1] G. McGraw, Software Security: Building Security In, Upper Saddle River, NJ: Addison-Wesley, 2006.

[2] R. Wettel, M. Lanza, and R. Robbes, “Software systems as cities: A controlled experiment,” in Proceedings of the 33rd International Conference on Software Engineering, ser. ICSE ’11. New York, NY, USA: ACM, 2011, pp. 551–560.

[3] OWASP.org, &quot;OWASP Top Ten Project,&quot; [Online]. Available:
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
[Accessed 12 December 2018].

[4] S. L. M. Howard, The Security Development Lifecycle, Redmond, WA: Microsoft Press, 2006.

[5] I. Arce, K. Clark-Fisher, N. Daswani, J. DelGrosso, D. Dhillon, C. Kern, T.
Kohno, C. Landwehr, G. McGraw, B. Schoenfield, M. Seltzer, D. Spinellis, I.
Tarandach and J. West, "Avoiding the top 10 software security design flaws,"
IEEE Computer Societys Center for Secure Design (CSD), 2014.

[6] L. R. Vanciu, "Static Extraction of Dataflow Communication for Security,"
Ph.D. dissertation, Wayne State University, Michigan, 2014.

[7] R. Kazman, "A tool to address cybersecurity vulnerabilites through design,"
SEI Blog, Software Engineering Institute, Carnegie Mellon University, 29
February 2016. [Online]. Available:
https://insights.sei.cmu.edu/sei_blog/2016/02/a-tool-to-address-cybersecurity-vulnerabilities-through-design.html. [Accessed 12 December 2018].

[8] R. Kissel, Glossary of Key Information Security Terms, United States:
National Institute of Standards and Technology (NIST), 2013.

[9]
R. S. Mark Sherman, "From Secure Coding to Secure Software," SEI Webinar Series, Software Engineering Institute, Carnegie Mellon University, 10 November 2016. [Online]. Available: https://www.sei.cmu.edu/webinars/view_webinar.cfm?webinarid=483646. [Accessed 12 December 2018].
[10] International Organization for Standardization, "ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models".

[11] CERT.org, "Cybersecurity Engineering," [Online]. Available: https://www.cert.org/cybersecurity-engineering/index.cfm#swamodel. [Accessed 12 December 2018].

[12] S. Lipner, "The trustworthy computing security development lifecycle," in Computer Security Applications Conference, 20th Annual, 2004.

[13] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Object-Oriented Reengineering Patterns. Morgan Kaufmann, 2002.

[14] Thomas Ball and Stephen G. Eick. Software visualization in the large. Computer, 29(4):33–43, 1996.

[15] Colin Ware. Information visualization: perception for design. Morgan Kaufmann Publishers Inc., 2 edition, 2004.

[16] Rainer Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. Journal of Software Maintenance, 15(2):87–109, 2003.]

[17] G. McGraw, J. Viega, G. Hoglund, G. McGraw, J. Viega and G. Hoglund, Software security library. [Upper Saddle River, N.J.]: Addison-Wesley, 2006.

[18] "Memo from Bill Gates - News Center", News Center, 2017. [Online]. Available:https://news.microsoft.com/2012/01/11/memo-from-bill-gates/#RUKBsWfxqsOIP1r3.97. [Accessed: 31- Jul- 2018].

[19] "At 10-Year Milestone, Microsoft's Trustworthy Computing Initiative More Important than Ever - News Center", News Center, 2017. [Online]. Available:https://news.microsoft.com/2012/01/12/at-10-year-milestone-microsoft's-trustworthy-computing-initiative-more-important-than-ever/#4xRiQOhjc5eAPPGD.97. [Accessed: 31- Jul- 2018].

[20]G. McGraw, &quot;Software security touchpoint: Architectural risk analysis,&quot; Technical report, Cigital, 2009.

[21]OWASP.org, &quot;Application Threat Modeling,&quot; [Online]. Available: https://www.owasp.org/index.php/Application_Threat_Modeling. [Accessed 18 December 2018].

[22]Microsoft, &quot;The STRIDE Threat Model,&quot; [Online]. Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. [Accessed 12 December 2018].

[23]M. Frydman, G. Ruiz, E. Heymann, E. César and B. Miller, &quot;Automating risk analysis of software design models.,&quot; The Scientific World Journal, 2014.

[24]X. Yuan, E. Nuakoh, J. Beal and H. Yu, &quot;Retrieving relevant CAPEC attack patterns for secure software development,&quot; in Proceedings of the 9th Annual Cyber and Information Security Research Conference, 2014.

[25]B. Berger, K. Sohr and R. Koschke, &quot;Automatically Extracting Threats from Extended Data Flow Diagrams,&quot; in International Symposium on Engineering Secure Software and Systems.

[26]Microsoft, &quot;SDL Threat Modeling Tool,&quot; [Online]. Available: https://www.microsoft.com/en-us/sdl/adopt/threatmodeling.aspx. [Accessed 12 December 2018].

[27]The ThreatModeler, &quot;Threat Modeling Tool,&quot; [Online]. Available: http://threatmodeler.com/threat-modeling-tool/. [Accessed 12 December 2018].

[28]I. Williams and X. Yuan, "Evaluating the effectiveness of Microsoft threat modeling tool," in Proceedings of the 2015 Information Security Curriculum Development Conference, 2015.

[29] H. Assal, "Collaborative security code review," in Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia, 2015.

[30] T. Ishikawa and K. Sakurai, "Parameter manipulation attack prevention and detection by using web application deception proxy," in Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, 2017.

[31] C. Heitzenrater, R. Böhme and A. Simpson, "The days before zero day: Investment models for secure software engineering," in Proceedings of the 15th Workshop on the Economics of Information Security (WEIS), 2016.

[32] N. Kaur and P. Kaur, "Mitigation of SQL injection attacks using threat modeling," ACM SIGSOFT Software Engineering Notes, 39(6), pp. 1-6, 2014.

[33] OWASP.org, "OWASP Proactive Controls," [Online]. Available: https://www.owasp.org/index.php/OWASP_Proactive_Controls. [Accessed 18 December 2018].

[34] Coverity, "Coverity White Paper: Building Security into Your Software Development Lifecycle," 2012.

[35] S. Alqahtani, E. Eghan and J. Rilling, "SV-AF—A Security Vulnerability Analysis Framework," in Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on, 2016.

[36] Find-sec-bugs.github.io, "Find Security Bugs," [Online]. Available: http://find-sec-bugs.github.io/. [Accessed 12 December 2018].

[37] sonarqube.org, "SonarQube," [Online]. Available: https://www.sonarqube.org/. [Accessed 12 December 2018].

[38] OWASP.org, "OWASP Top Ten Project," [Online]. Available:
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
[Accessed 12 December 2018].

[39] R.Wettel and M.Lanza. - Visualizing Software Systems as Cities. In VISSOFT
[40] Using High-Rising Cities to Visualize Performance in Real-Time"
https://ieeexplore.ieee.org/document/8091184/

[41]"Representing Development History in Software Cities" http://dl.acm.org/citation.cfm?id=1879239

[42] A complete 3D game development suite written purely in Java.
https://github.com/jMonkeyEngine/jmonkeyengine

[43]S. C. Eick, J. L. Steffen, and E. E. Sumner, "Seesoft-a tool for visualizing line oriented software statistics," IEEE Transactions on Software Engineering, vol. 18, pp. 957–968, Nov 1992.

[44] A. Marcus, L. Feng, and J. I. Maletic, "3d representations for software visualization," in Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis '03, (New York, NY, USA), pp. 27–ff, ACM, 2003.

[45] A. Goldberg and D. Robson, Smalltalk-80: The Language and Its Implementation. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983.

[46] C. Kurtz, "Code gestalt: A software visualization tool for human beings," in CHI '11 Extended Abstracts on Human Factors in Computing Systems, CHI EA '11, (New York, NY, USA), pp. 929–934, ACM, 2011.

[47] M. Lanza and S. Ducasse, "A categorization of classes based on the visualization of their internal structure: The class blueprint," SIGPLAN Not., vol. 36, pp. 300–311, Oct. 2001.

[48]C. Gutwenger, M. J¨unger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel, "A new approach for visualizing uml class diagrams," in Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis '03, (New York, NY, USA), pp. 179–188, ACM, 2003.

[49]M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, (New York, NY, USA), pp. 165–172, ACM, 2005.

[50] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," IEEE Transactions on Visualization and Computer Graphics, vol. 12, pp. 741–748, Sept 2006.

[51] N. Hawes, S. Marshall, and C. Anslow, "Code Surveyor: Mapping large-scale software to aid

in code comprehension," in 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), pp. 96–105, Sept 2015.

[52] A. Cockburn and B. McKenzie, "Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '02, (New York, NY, USA), pp. 203–210, ACM, 2002.

[53] A. Cockburn and B. McKenzie, "Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '02, (New York, NY, USA), pp. 203–210, ACM, 2002.

[54] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich, "Data mountain: Using spatial memory for document management," in Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, UIST '98, (New York, NY, USA), pp. 153–162, ACM, 1998.

[55]D. Saito, H. Washizaki, and Y. Fukazawa, "Influence of the programming environment on programming education," in Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '16, (New York, NY, USA), pp. 354–354, ACM, 2016.

[56] Definition of Parser. https://searchmicroservices.techtarget.com/definition/parser

Appendices

Appendix A : Individual Contribution

Member 1: A.A.T.G.Abeysinghe
A comparison of the existing static code analysis tools was conducted to select the appropriate static code analysis tool to be used with the identified approach. The work related in implementing the Spring Boot API using a Java parser, in order to pass the necessary source code information from the First Level view to Second Level View was carried out. The implementation of the Third Level view, IDE navigation and issue posting mechanism to Trello were conducted.

Member 2: M.A.S.Shalika
A Comparison of the Existing Software Visualization Techniques Was Conducted to find a appropriate visualization for mehid level.A visualization mechanism for metrics of methods inside software project and a visualization mechanism for vulnerabilities in methods was done.Coloring mechanism was introduced to assign severity level for each method inside the class.Three graphs which depict information regarding some important aspects related to software security, were developed,which are vulnerability distribution in the class, displaying percentage of severity levels of SonarQube and number of OWASP issues in particular class. Filtering mechanism was done to obtain methods which include selected OWASP vulnerabilities.

Member 3: M.S.N. Ahamed

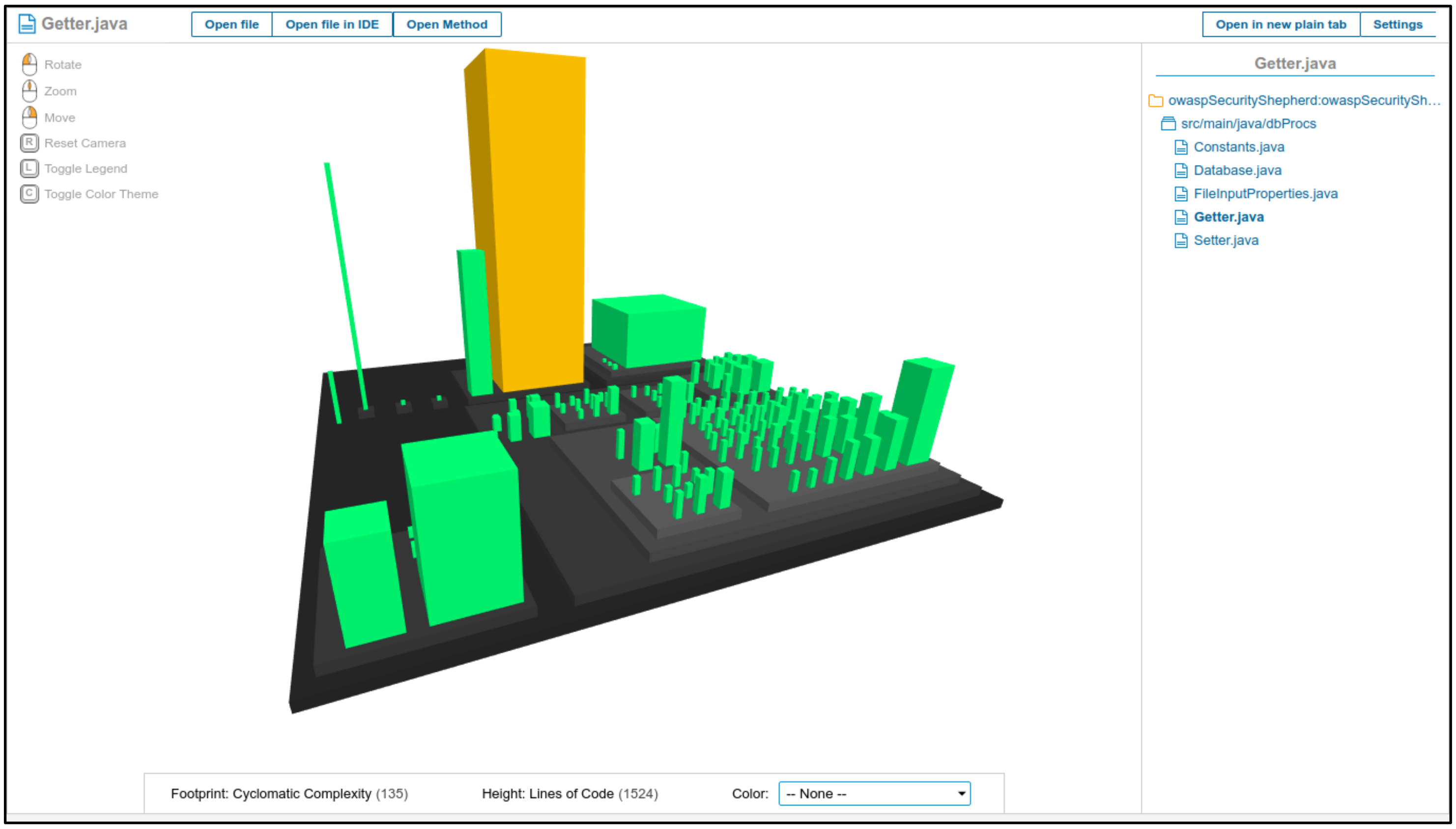
Literature Review and Comparison on Various Software Visualization Technique and Source Code analysis tools were conducted in order to get most suitable tool to get Project Measures and Security Vulnerability Details for Visualizing First Level View.An algorithm was implemented to build the Secure CodeCity and Color the Building according to Security Vulnerability Severity, Security Vulnerability Rating, Security Remediation Effort and Cognitive Complexity of the classes, which was built using React, node JS and TypeScript with Unit test cases. Also implemented First Level View to show above mentioned Measure once building was selected, And built a mechanism to Navigate to particular file in IDE. Contributed to Designing and Modelling of System Architecture, Designing and Implementation of Components of First Level View. Contributed to Designing and Implementation of the File Hierarchy and coloring building according to number of author.

Member 4: S.M.Mufarrij

A Comparison of the Existing Software Visualization Techniques was Conducted In Order to Come Up with the Appropriate Visualization Methodology for First Level Visualization of SecureCode City.Works related to Designing and Modelling of System Architecture, Designing and Implementation of Components of First Level View was carried out.Designing and Implementing *“**Visualization Helper**”* Component For the System Which Includes Algorithms and Layouts for Visualization of The City. Contributed to Frontend Designing and UI Development of the First Level View,Writing Unit Test cases for First Level Visualization Module.Also Contributed to Design and Implementation of FileHireracy and Coloring Buildings based on number of Authors.Implementing Navigation from IDE to the System was done.**System Integration** was performed by Integrating First Level Visualization ,Second Level Visualization and Third Level Visualization Sub Systems of the Entire System.Also Contributed to Second Level and Third Level Visualization Subsystems Development as well.

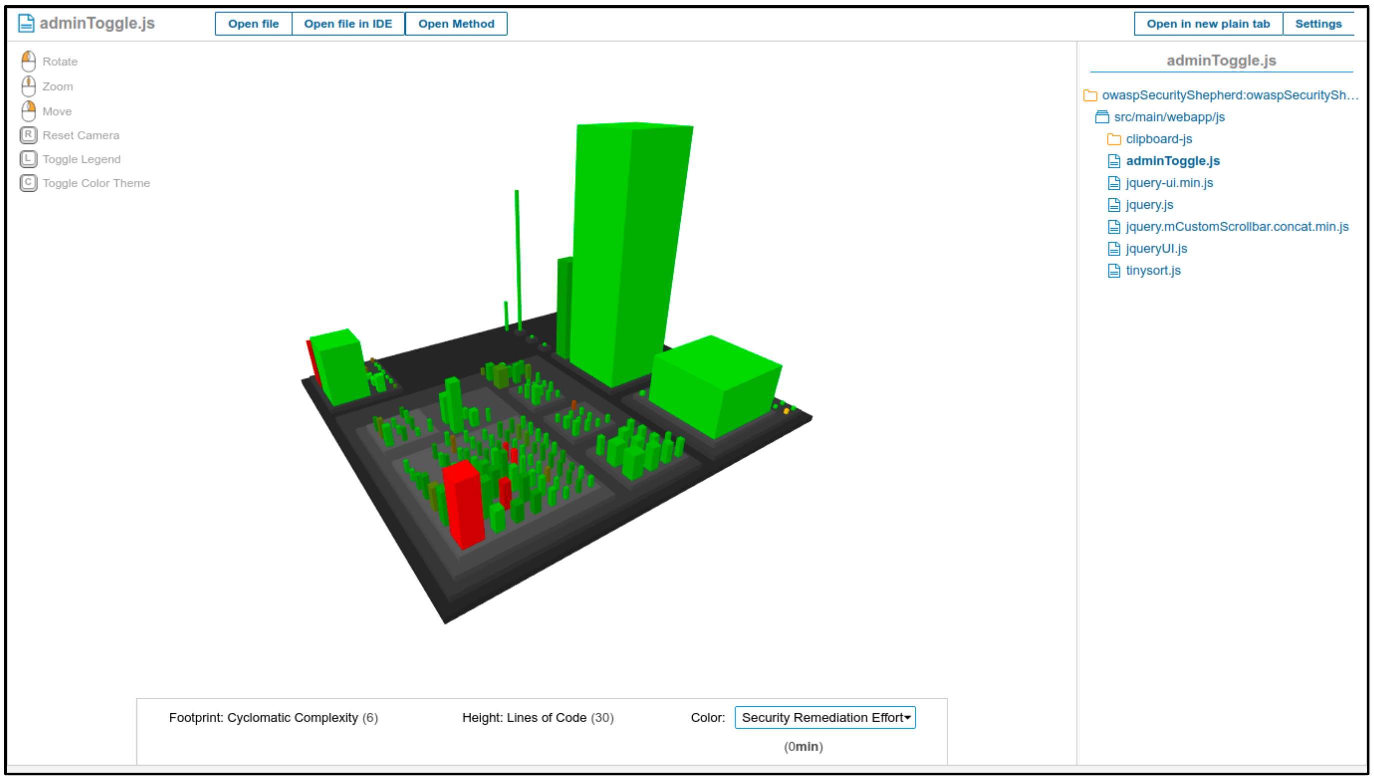
Appendix B : User Interfaces

Figure below : Default View (First Layer View)

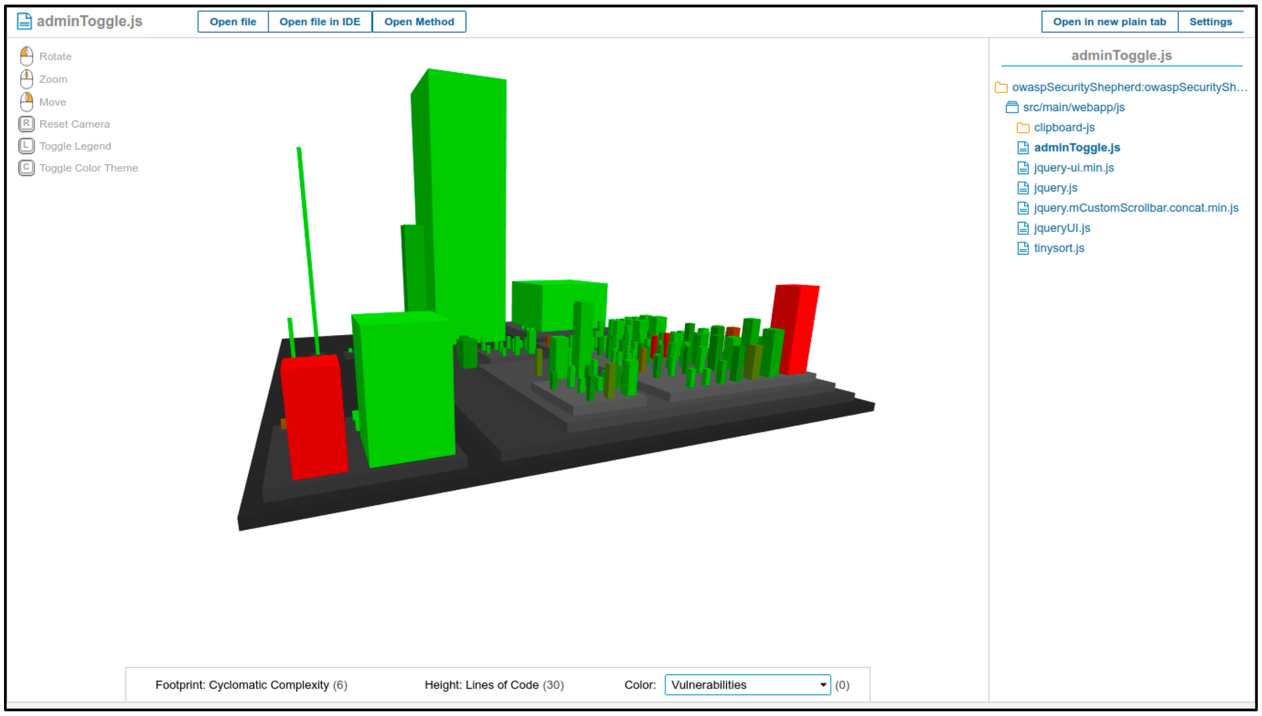


Footprint: Cyclomatic Complexity (135) Height: Lines of Code (1524) Color: -- None --

Figure Below : Distribution of color according to Security Remediation Effort (Green : low security remediation effort, Red : High security remediation effort). Green to Red spread, sonarqube standard color



Footprint: Cyclomatic Complexity (6) Height: Lines of Code (30) Color: Security Remediation Effort (0min)



Footprint: Cyclomatic Complexity (6) Height: Lines of Code (30) Color: Vulnerabilities (0)

Figure above : shows the security vulnerability distribution (Less vulnerable class : Green Building, High Vulnerable class : Red Color). Selected Building colored with yellow, Class name is shown in Top left corner. Selected Building Cyclomatic Complexity, Line of Code , Vulnerability values are shown in label below the Secure CodeCity.

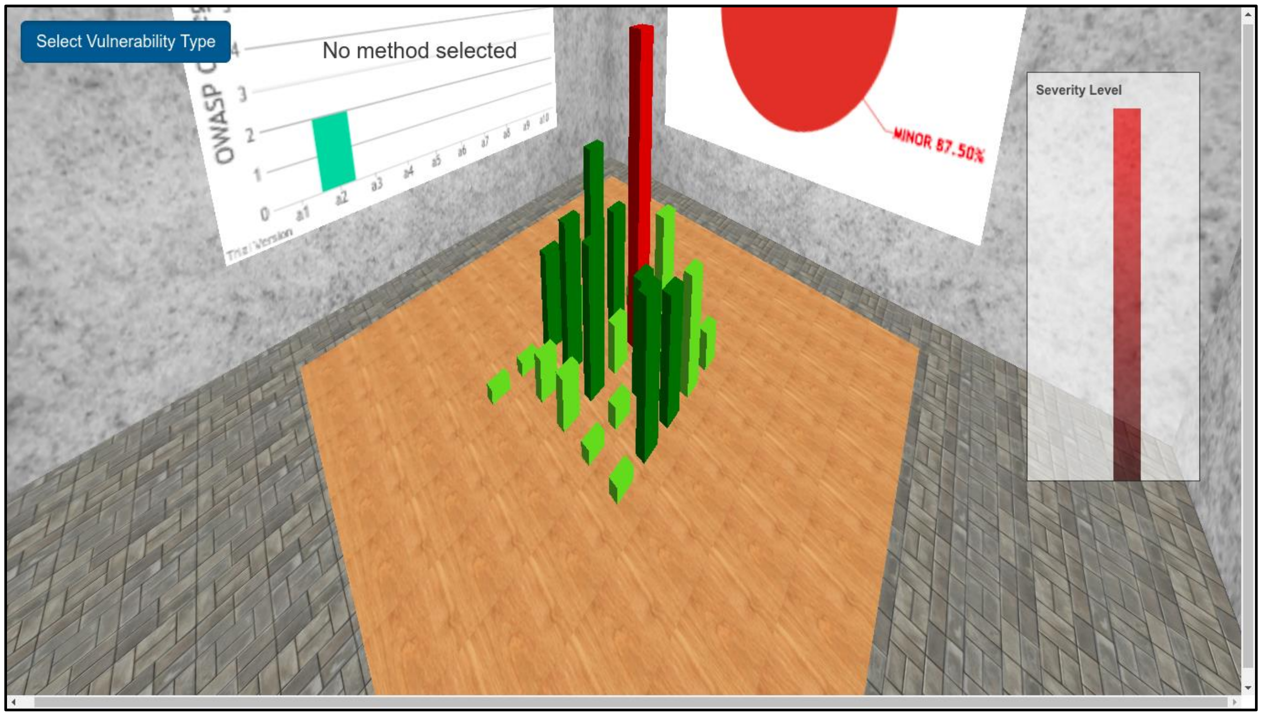


Figure above : Default View (Second Layer View)

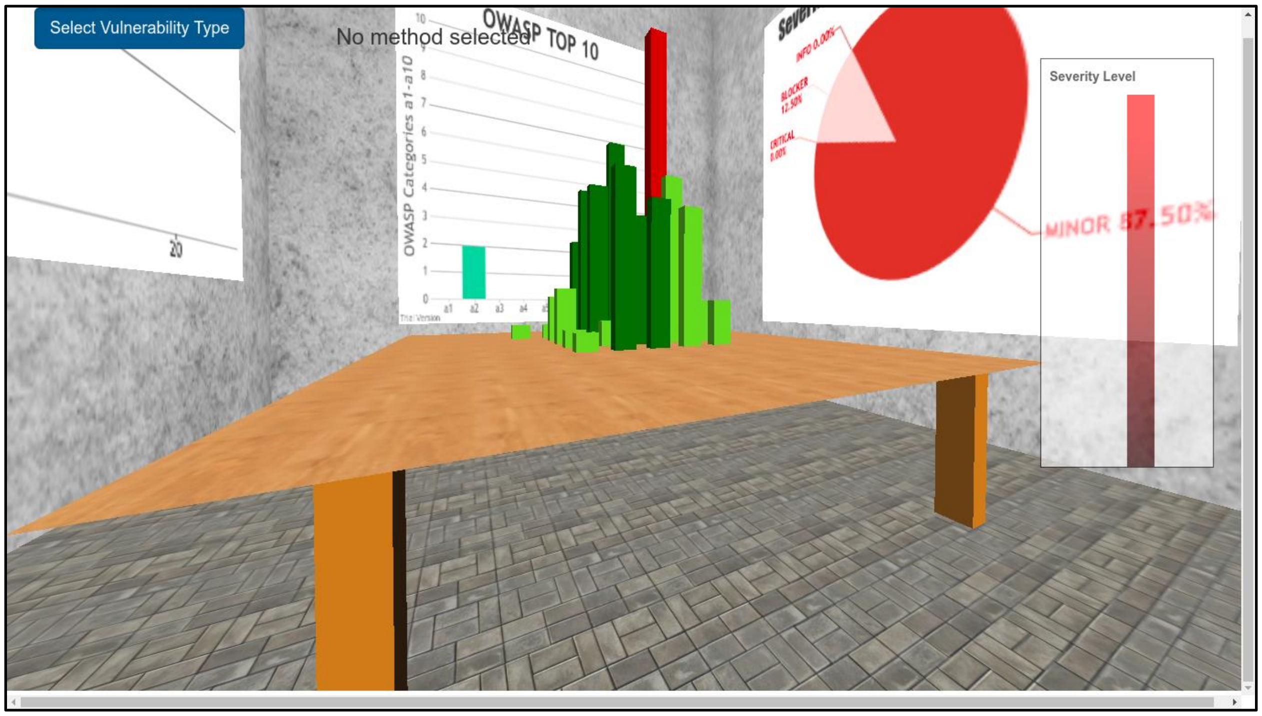


Figure above : One wall displays OWASP issues chart.

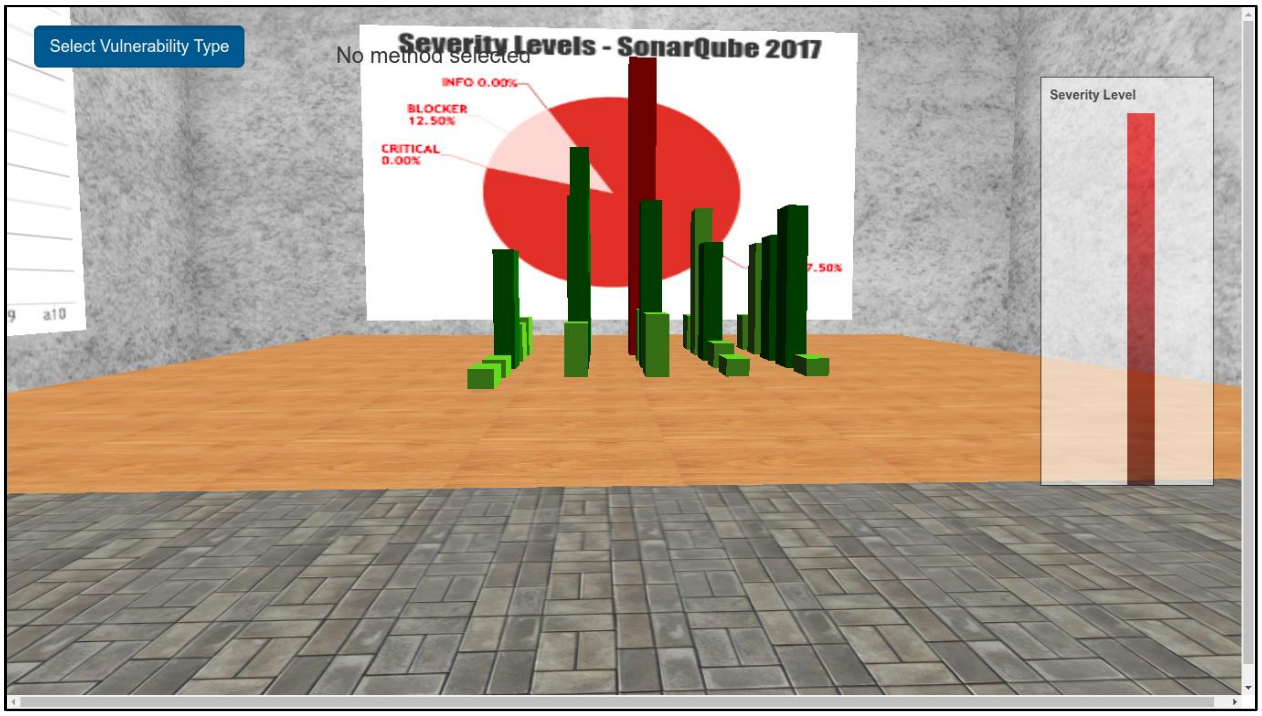


Figure above : One wall displays percentage chart of security levels of SonarQube.

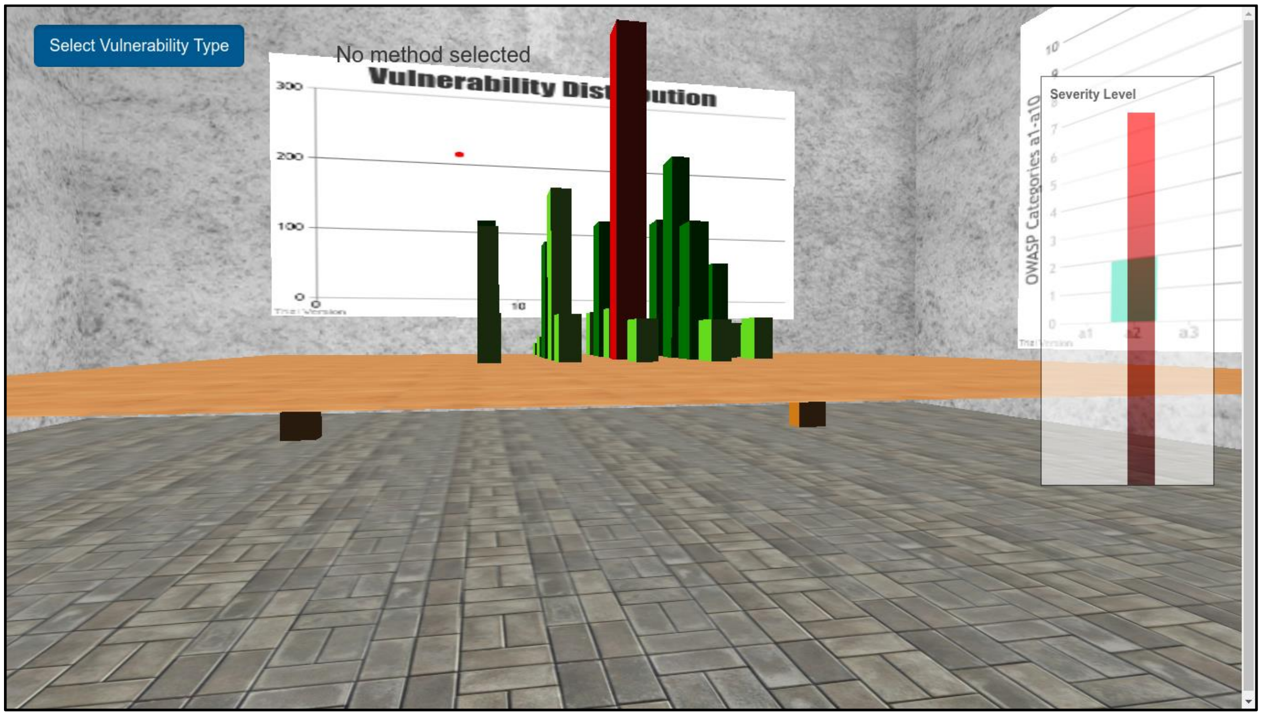
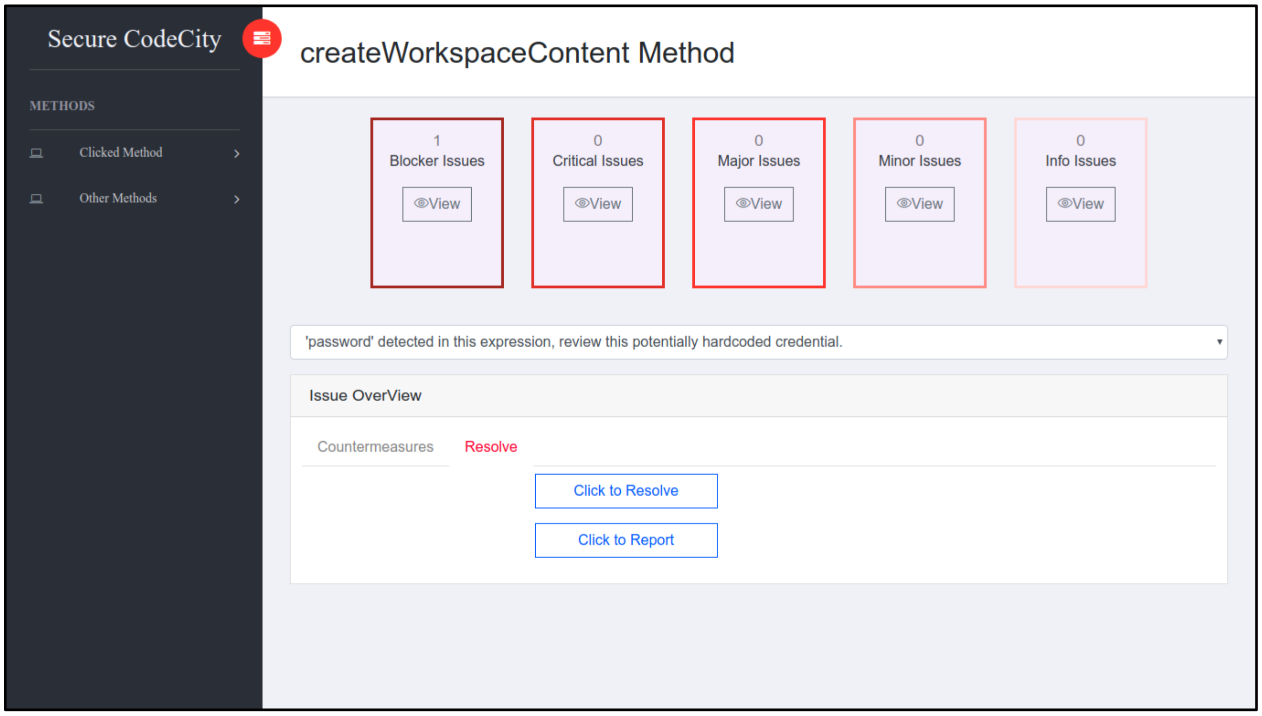
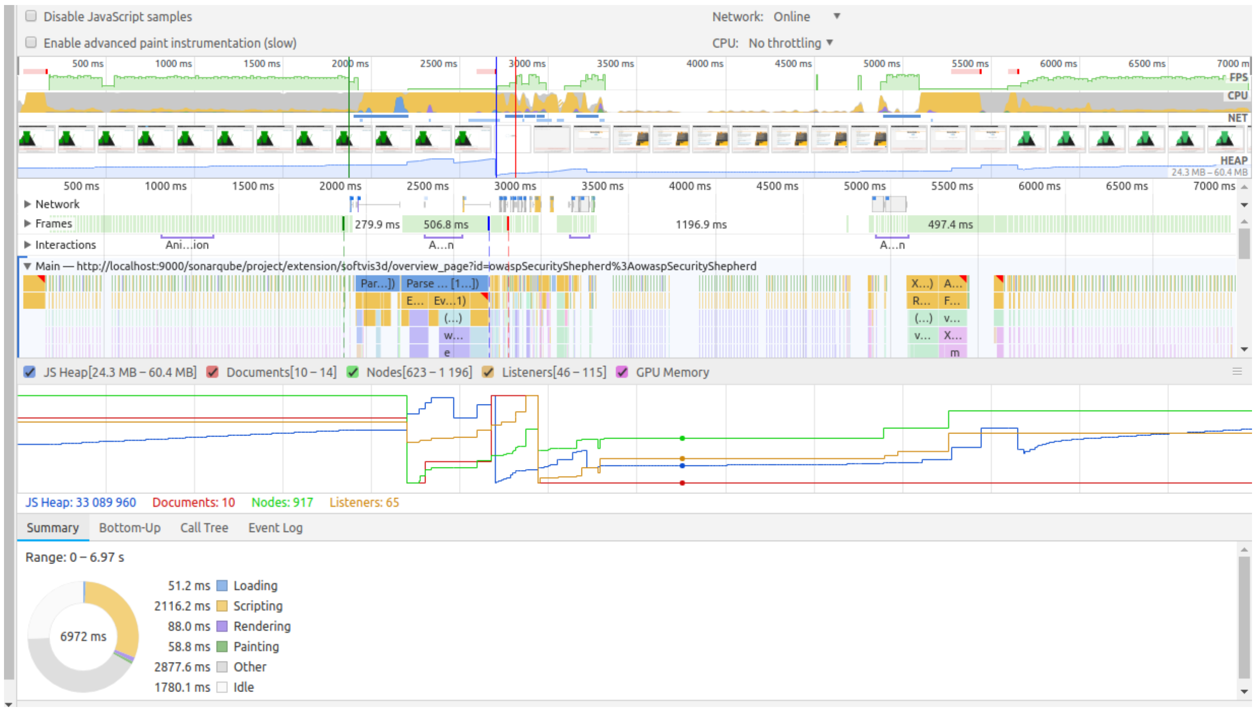


Figure above : One wall displays vulnerability distribution inside the class.



'password' detected in this expression, review this potentially hardcoded credential.

Figure above : Default View (Third Layer View)



☐ Disable JavaScript samples Network: Online

☐ Enable advanced paint instrumentation (slow) CPU: No throttling

Main — http://localhost:9000/sonarqube/project/extension/softvis3d/overview_page?id=owaspSecurityShepherd%3AowaspSecurityShepherd

✓ JS Heap[24.3 MB – 60.4 MB] ✓ Documents[10 – 14] ✓ Nodes[623 – 1 196] ✓ Listeners[46 – 115] ✓ GPU Memory

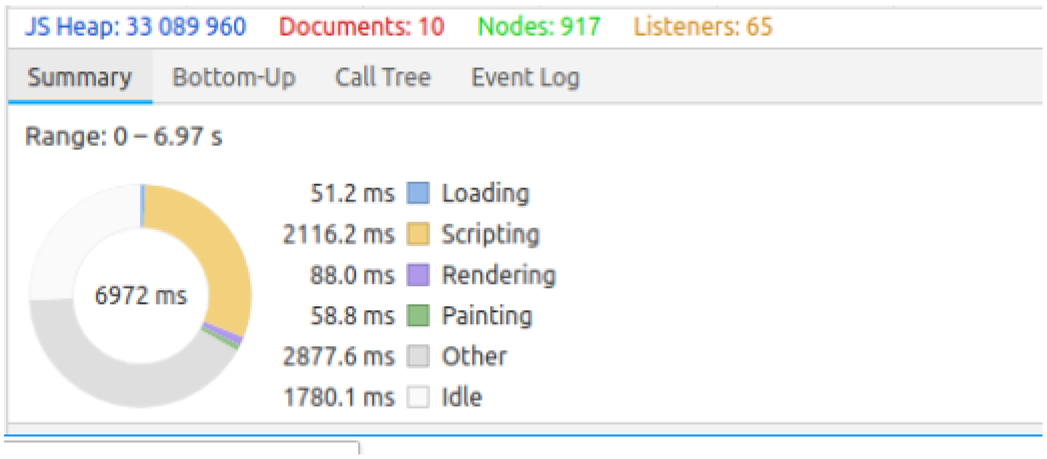
JS Heap: 33 089 960 Documents: 10 Nodes: 917 Listeners: 65

Summary Bottom-Up Call Tree Event Log

Range: 0 – 6.97 s

6972 ms

51.2 ms	🟦	Loading
2116.2 ms	🟨	Scripting
88.0 ms	🟪	Rendering
58.8 ms	🟩	Painting
2877.6 ms	⬜	Other
1780.1 ms	⬜	Idle


JS Heap: 33 089 960 Documents: 10 Nodes: 917 Listeners: 65

Summary Bottom-Up Call Tree Event Log

Range: 0 – 6.97 s

6972 ms

51.2 ms 🟦 Loading
2116.2 ms 🟨 Scripting
88.0 ms 🟪 Rendering
58.8 ms 🟩 Painting
2877.6 ms ⬜ Other
1780.1 ms ⬜ Idle

Figure above : Profiling of Secure CodeCity Initial loading

Appendix C : Use Case

Appendix D : Evaluation Results

D(1)

Correctness Values of Each Subject against the Task List - Experimental Group

Secure CodeCity (experimental group) - Task results for Correctness Evaluation												
Task No.	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Subj 7	Subj 8	Subj 9	Subj 10	Subj 11	Total
1	T	T	T	T	T	T	T	T	T	T	T	11
2	T	timeout	T	T	T	T	T	T	T	T	T	10
3	T	T	T	T	T	T	T	T	T	T	T	11
4	T	T	T	T	T	timeout	T	T	T	T	T	10
5	T	T	T	T	T	T	T	T	T	T	T	11
6	T	T	T	T	T	T	T	T	T	T	T	11
7	T	T	T	T	T	T	T	T	T	T	T	11
8	T	F	T	T	T	T	T	T	T	T	T	10
9	T	T	T	T	T	T	T	T	T	T	T	11
10	T	T	T	T	T	T	T	T	T	T	T	11
11	T	T	T	T	T	T	T	T	T	T	T	11

D(2)

Correctness Scores of the Tasks - Experimental Group

	Secure CodeCity (experimental group)-Correctness Scores										
Task	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Score	11	10	11	10	11	11	11	10	11	11	11

D(3)

Correctness Values of Each Subject against the Task List - Experimental Group

SonarQube(control group)	

Task No.	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Subj 7	Subj 8	Subj 9	Subj 10	Subj 11	Total
1	timeout	timeout	F	T	T	time out	T	F	F	timeout	T	4
2	T	T	T	timeout	T	T	F	T	T	T	T	9
3	T	timeout	T	timeout	T	F	T	timeout	T	T	F	6
4	T	T	T	T	T	T	T	T	T	T	T	11
5	T	T	T	T	T	F	T	T	T	T	T	10
6	T	F	timeout	F	F	T	time out	F	T	F	T	4
7	T	F	T	T	T	T	F	T	T	T	T	9
8	T	T	T	T	T	T	T	T	T	T	T	11
9	T	F	T	T	T	T	T	T	T	T	T	10
10	T	T	T	T	T	T	T	T	T	T	T	11
11	T	timeout	T	T	T	T	T	T	T	T	T	10

D(4)

Correctness Scores of the Tasks - Control Group

	SonarQube (control group)-Correctness Scores

Task	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Score	4	9	6	11	10	4	9	11	10	11	10

D(5)

CSUQ Scores of the Subjects - Experimental Group

	Secure CodeCity (experimental group)Usability Scores										
Subject	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Subj 7	Subj 8	Subj 9	Subj 10	Subj 11
CSUQ Score	92.7536	89.8550	97.1014	87.5362	93.5456	84.3124	79.7101	95.3384	81.3216	78.5413	90.577

D(6)

CSUQ Scores of the Subjects - Control Group

	SonarQube(control group)Usability Scores

Subject	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Subj 7	Subj 8	Subj 9	Subj 10	Subj 11
CSUQ Score	77.6541	73.1884	85.3343	90.1367	72.1554	62.2136	74.9651	74.3125	78.5418	93.2401	67.1489

D(7)

Overall Completion Time of the Tasks for Each Subject - Control Group

SonarQube(control group) -time efficiency		

Task No.	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Subj 7	Subj 8	Subj 9	Subj 10	Subj 11	Total	AVG
1	600s (timeout)	600s (timeout)	195s	490s	540s	600s(timeout)	375s	120s	340s	600s(timeout)	395s	4855	441.36
2	20s	35s	32s	600s (timeout)	60s	54s	45s	70s	34s	28s	25s	1003	91.18
3	554s	600s (timeout)	360s	600s (timeout)	435s	300s	500s	600s (timeout)	405s	515s	596s	5465	496.82
4	60 s	230s	55s	45s	50s	45s	40s	45s	55s	70s	80s	775	70.45
5	155s	240s	150s	120s	200s	175s	180s	143s	102s	240s	152s	1817	165.18
6	300s	420s	600s(timeout)	240s	240s	315s	600s(timeout)	244s	300s	285s	297s	3571	324.64
7	40s	50s	25s	30s	35s	35s	28s	30s	40s	35s	35s	383	34.82
8	44s	110s	50s	45s	45s	48s	47s	52s	45s	53s	49s	588	53.45
9	98s	180s	135s	100s	120s	165s	172s	110s	130s	115s	100s	1425	129.55
10	160s	140s	163 s	155 s	165s	157s	118s	130s	144s	115s	120s	1567	142.45
11	156s	600s (timeout)	160s	300s	140s	165s	310s	153s	170s	162 s	178s	2494	226.73
Total	2187s 36.45min	3205 53.41	1925 32.08	2725 45.42	2030 33.83	2059 34.32	2415 40.25	1697 28.28	1765 29.42	2218 36.97	2027 33.78		

D(8)

Overall Completion Time of the Tasks for Each Subject - Experimental Group

SCC(experimental group) - time efficiency													
Task No.	Subj 1	Subj 2	Subj 3	Subj 4	Subj 5	Subj 6	Sub j 7	Subj 8	Subj 9	Subj 10	Subj 11	Total	AVG
1	5s	7s	15 s	10s	10s	15s	8s	7s	15s	10s	10s	112	10.18
2	60s	600s (tim	32s	40s	20s	22s	30s	28s	20s	25s	25s	902	82

		eout)											
3	8s	15s	10s	15s	22s	20s	15s	20s	10s	12s	15s	162	14.73
4	52s	60s	35s	45s	20s	600 s(ti meo ut)	30s	20s	15s	30s	34s	941	85.55
5	40s	20s	30s	20s	25s	45s	30s	23s	26s	30s	22s	311	28.27
6	10s	12s	15s	11s	20s	10s	8s	14s	20s	10s	10s	140	12.73
7	8s	10s	24s	10s	15s	25s	20s	10s	12s	15s	18s	167	15.18
8	14s	10s	22s	10s	15s	15s	14s	12s	10s	35s	10s	167	15.18
9	10s	14s	15s	26s	15s	40s	10s	12s	15s	11s	8s	176	16n
10	10s	30s	12 s	8 s	15s	10s	13s	10s	14s	10s	10s	142	12.91
11	6 s	8s	5s	5s	6s	5s	5s	8s	5s	5s	7s	65	5.91
Tota l	223 3.71 min	786 13.1	215 3.58	200 33.3 3	173 2.88	807 13.4 5	183 3.0 5	164 2.73	162 2.7	193 3.22	169 2.82		

Appendix E : Terminology

1. Vulnerability - An unexpected and relatively small defect, fault, flaw, or imperfection in an information system or device.

2. Build Security In - A set of principles, practices, and tools to design, develop, and evolve information systems and software that enhance resistance to vulnerabilities, flaws, and attacks.

3. Defect - A problem that may lie dormant in software for years only to surface in a fielded system with major consequences.

4. Exploit - A script or plan that executes against a vulnerability, leading to a security compromise.

5. Risk - The potential for an unwanted or adverse outcome resulting from an incident, event, or occurrence, as determined by the likelihood that a particular threat will exploit a particular vulnerability, with the associated consequences.

6. Flaw - A design-level or architecture-level software defect.

7. Software Security - The idea of engineering software so that it continues to function correctly under malicious attacks.

8. Threat - A circumstance or event that has or indicates the potential to exploit vulnerabilities and to adversely impact organizational operations, organizational assets, individuals, other organizations, or society.

9. Touchpoint - A characteristic or specific weakness that renders an organization or asset open to exploitation by a given threat or susceptible to a given hazard.

10. Vulnerability - A characteristic or specific weakness that renders an organization or asset (such as information or an information system) open to exploitation by a given threat or susceptible to a given hazard.

11. React : A declarative, efficient, and flexible JavaScript library for building user interfaces which was built by Facebook.It is used to compose complex UIs from small isolated components.

12. MobX : MobX is a battle tested library that makes state management simple and scalable by transparently applying functional reactive programming (TFRP).

13. Sass :Sass is an extension of CSS that adds power and elegance to the basic language. It allows you to use variables, nested rules, mixins, inline imports, and more, all with a fully

CSS-compatible syntax. Sass helps keep large stylesheets well-organized, and get small stylesheets up and running quickly, particularly with the help of the Compass style library.

14. Enzyme : Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.

15. Chai : Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

16. Mocha : Mocha is a feature-rich JavaScript test framework running on Node.js and in the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub

17. Sinon : Standalone test spies, stubs and mocks for JavaScript. Works with any unit testing framework.

18. Convas-js : CanvasJS is an easy to use JavaScript & HTML5 Charts library built on Canvas element.This allows you to create rich dashboards that work on all the devices without compromising on maintainability or functionality of your web application