2016

An information sharing system for crowd-sourced software testers

Otolo, R. A.
Faculty of Information Technology (FIT)
Strathmore University

An Information Sharing System for Crowd-sourced Software Testers

Richard Assanga Otolo

079249

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Mobile Telecommunications and Innovation at Strathmore University

Faculty of Information Technology

Strathmore University

Nairobi, Kenya

June 2016

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, this dissertation contains no material previously published or written by another person except where due reference is made in the dissertation itself.

Otolo Richard Assanga 079249

Signature _____

Date _____16th June, 2016 _____

Approval

The dissertation of Richard Assanga Otolo was reviewed and approved by the following

Dr. Humphrey Njogu

Lecturer, Faculty of Information Technology

Strathmore University

Dr. Joseph Orero,

Dean, Faculty of Information Technology

Strathmore University

Prof. Ruth Kiraka,

Dean, School of Graduate Studies

Strathmore University

Dedication

This work is dedicated to my mother, Mrs. Margaret Otolo without whom this whole journey would not have been possible. Thank you for your unfailing support, encouragement and constant prayers. We made it!

ACKNOWLEDGEMENTS

I wish to thank Dr. Humphrey Njogu, for his excellent supervision and guidance. Thank you for always making time to see me and review my work. You never missed an appointment! Thank you.

I would like to thank everyone at Fintech who assisted with making this study a success.

I would like to thank all my friends at Trinity Chapel Ruiru and Ignite for their patience, encouragement and prayers over the past few months.

ABSTRACT

Consumer demand for mobile applications is on an upward trend. This demand has resulted in increasing pressure on software developers to develop and deliver these applications within tight deadlines. Moreover, these applications still need to be tested to ensure they are secure. This need to deliver well-tested applications within the shortest timeframe possible has led developers to consider using crowd-sourced software testing platforms.

These platforms offer access to a large pool of testers who can test software faster and more effectively than traditional in-house testing. However, these systems come with their own challenges particularly in how test-related information is shared. They generally do not provide real-time communication between testers and developers, they also do not send real-time alerts to users in the system. These challenges reduce their effectiveness as a possible avenue of testing software.

As the cost of hardware and cell phone usage continues to decrease, mobile phones present a unique opportunity to provide a channel that addresses the communication channels on crowd-sourced software testing platforms.

In this study, a prototype system is developed that improves the delivery of communication between software developers and testers on crowd-sourced software testing platforms. The system improves communication between testers and developers by providing modules that allow for sharing of test-related data in real-time.

A select sample of software developers in Nairobi County was chosen to test the application. The respondents indicated the features they found most useful were the ability to receive real-time alerts of new projects as well as the ability to chat in real-time with other testers on the platform.

Keywords: *Crowdsourced software testing, crowdsourcing*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application programming interfaces
CAPEC	Common Pattern Enumeration and Classification
CSS	Cascading Style Sheets
CSV	Comma Separated Value
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
MDP	Metrics Database Program
MFSA	Mozilla Foundation Security Advisories
MSRC	Microsoft Security Response Center
NASA	National Aeronautics and Space Administration
NVD	National Vulnerability Database
OSDVB	Open Source Vulnerability Database
SDL	Security Development Lifecycle
SDLC	Software Development Lifecycle
UML	Unified Modeling Language

CHAPTER 1: INTRODUCTION

1.1 Background

The demand for mobile applications is growing quickly. Research carried out by Portio reports that by the end of 2012, 46 billion mobile applications had been downloaded from application marketplaces. This was projected to grow to 82 billion applications by the end of 2013 (Whitfield, 2013). To meet this high demand for mobile applications developers need to develop and release applications as fast as possible. However, research suggests that the developers may not be adequately testing the applications they develop for security vulnerabilities.

In 2012, the Ponemon Institute surveyed a population consisting of developers and application security testers. The results indicated that 44% of the developers did not involve the application security team when they were testing their applications. Additionally, over 50% of both the developers and testers said they lacked formal training in basic application security. Thus, they never tested the applications they developed for security weaknesses (Ponemon Institute, 2012). To achieve the dual goal of delivering secure applications quickly, new approaches to development and testing are needed.

Crowd-sourcing is defined as "the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call." (Shen, 2015). When software testing is outsourced it is termed crowd-sourced testing. Crowd-sourced testing offers several advantages to developers. They offer access to a large pool of highly skilled application security testers who because of their numbers and diversity are able to offer better testing within a shorter timeframe than traditional testing. These crowdsourced testers usually have better resources, better skills, can perform faster testing and are more affordable than in-house testers (Mao, Capra, Harman, & Jia, 2015). However, this approach also has its share of challenges.

1.2 Problem Statement

Crowd-sourced testing works by allowing testers and developers to connect via an Internet-based testing platform provided by a crowdsourcing company. Existing platforms have several gaps in the way sharing of test-related information among different stakeholders is handled.

These platforms generally do not offer developers and testers a way of sharing test-related information in real-time and on demand. One outcome of this is that testers are unable to consult with each other during testing. This leads to less that optimum test results. Furthermore, the platforms lack functionality that sends real-time alerts to testers. As a result of this, two or more testers end up reporting on the same vulnerability simultaneously because they lack visibility on what is being reported.

1.3 Research Objectives

This research is based on the following objectives:

a) To investigate the sources of security vulnerabilities in software applications.
b) To identify the gaps that existing crowd-sourced testing platforms have in terms of sharing of test-related information.
c) To design and develop a mobile-based prototype for use by crowd-sourced testers and developers to share test-related information.
d) To evaluate the effectiveness of the developed prototype in improving the way test-related information is shared.

1.4 Research Questions

This research will seek to answer the following research questions

a) What are the sources of security vulnerabilities in software applications?
b) What gaps do existing crowd-sourced application testing systems have in terms of sharing test-related information?
c) How can a prototype that addresses information sharing gaps in existing crowd-sourced software testing platforms be designed?
d) How effective is the developed prototype in improving the sharing of test-related information?

1.5 Significance of the Study

This study seeks to develop a system that will improve the speed and accuracy of identifying and reporting of test-related information on crowdsourced software testing platforms. It does this by facilitating real-time, on-demand communication between

developers and testers. This will enable clarification, real-time alerts, and accurate information to be shared during testing projects. This will lead to more secure applications being developed and released to the market at a fast rate.

1.6 Scope and Limitations of the Study

Software testing is wide and the application of technology requires a broad and varied approach. This study is limited to developing a mobile application that improves communication between developers and testers working on crowd-sourced software testing platforms. The software will be tested for security issues and the various stakeholders are expected to have and be conversant with using android-based smartphones. Lastly, the research will be done in English

1.7 Assumptions of the Study

The study assumes that the testers and developers in the study are from an urban area which in this case is Nairobi County. The county was chosen because of the higher likelihood of finding respondents with adequate background and exposure to crowd-sourced software testing platforms. Furthermore, the majority of the people in this county are considered to have access to and be conversant with using smartphones.

1.8 Structure of the Dissertation

The rest of this work is organized as follows: Chapter 2 is a review of the literature. Chapter 3 discusses the evolutionary prototyping method. Chapter 4 presents the design and architecture of the proposed system. Chapter 5 discusses how the system will be implemented and tested. Chapter 6 provides an evaluation of the test results. Chapter 7 provides the conclusion to this dissertation provides recommendations for future work.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

This literature review begins with a general overview of application security in the software development lifecycle. It then focusses on secure software development methodologies. Next vulnerability management systems are discussed followed by an analysis of how software testing is conducted in crowd-sourced systems. The chapter then ends with a review of existing crowd-sourced software testing systems, a conceptual framework, and the conclusion.

2.2 Application Security in the Software Development Lifecycle

Quality software is that which is delivered on time, within budget and satisfies the desired requirement (Thakral & Sharma, 2014). One way to achieve this goal is by the use of sound engineering principles which are based on a process framework. Generally, the framework consists of five processes namely: Communication, Planning, Modelling, Construction and Deployment. To aid software managers to monitor the progress of a project, a number of models have been proposed based on this framework. Each time software is developed it undergoes a series of special stages known as the Software Development Lifecycle (SDLC). A software lifecycle model is a diagrammatic representation of the software lifecycle (Thakral & Sharma, 2014).

The communication phase establishes the business need based on a preliminary list of customer requirements. It is at this stage that the product and a strategy for the road map for the next stage of the lifecycle are decided. The planning phase involves deciding on such vital items as the number of resources required based on a detailed market, customer and financial assessment. The construction phase has the following sub-phases: requirements, design, implementation and test phases. This stage is where most of the security is usually introduced well as most of the vulnerabilities. The deployment phase is where thousands of copies of the software are made available for distribution (Gupta, Chandrashekhar, Sabnis, & Bastry, 2007).

The most familiar software development model is the waterfall model shown in figure 2.1. This model has several drawbacks which include escalating, unsatisfactory reliability, performance, and functionality of the resulting software (Davis, Bersoff, &

Comer, 1988). In support of this findings, Thakral & Sharma (2014) point out that the waterfall model assumes that no development error is ever committed in previous stages. However, practically speaking, each phase of the waterfall model tends to have a large number of errors which spill over to the next stage. This is because there is no mechanism to handle these errors. The sources these defects include oversight, communication gap, use of the wrong software etc. These shortcomings have led software engineers to propose different types of software development models to address these weaknesses

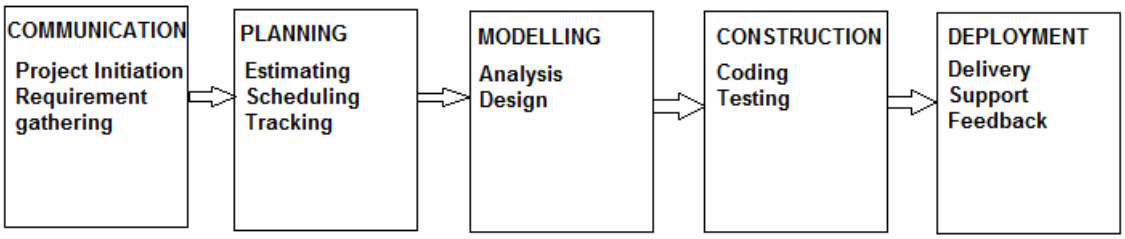

Figure 2.1 Waterfall model. Source (Thakral & Sharma, 2014, 73)

Over time, many Software Development Lifecycle (SDLC) models have been proposed and they can be classified into three major groups: linear, iterative and a combination of the two (Ruparelia, 2010). A linear model is executed in a sequential manner whereby the next stage can only start once the previous stage is complete, such as in the waterfall model. An iterative model requires that all the stages are reassessed at some point in the future for the purposes of continuous improvement. A combined model requires that the iterative approach is stopped at a certain stage. Examples of sequential models are the waterfall model, the B-model, the V-Model. Examples of iterative models are Unified process model, Rapid application development, Agile, Scrum, Extreme programming, Joint application development, Lean development. Examples of combined models include the Spiral model, and the Wheel and spoke model.

It is helpful to use common frameworks to guide process implementation as well as to evaluate processes against a common model to determine areas for improvement. However, very few of these models were initially designed with the aim of addressing security (Noopur, 2005). Thus, developers need to shift their mindsets when it comes to developing secure software. Developers need to not only think that their software will solve problems but that it will also be attacked. They thus must consider software security as an integral part of the development process.

2.3 Secure Software Development Lifecycle

2.3.1 Vulnerabilities and the SDLC

A vulnerability is "an instance of a [fault] in the specification, development, or configuration of software such that its execution can violate the [implicit or explicit] security policy." (Krsul, 1998). The listing below describes how vulnerabilities tend to manifest in the various phases of the SDLC.

1. **Requirements definition phase**

   Vulnerabilities that occur here are a result of inadequate or often completely absent security-centric requirements. Requirements may fail to consider unintended and malicious users. Security requirements are often stated as nonfunctional requirements, specifying a particular technology to use rather than identifying threats and characterizing the threat environment (Sindre & Opdahl, 2008).

2. **Design phase**

   Vulnerabilities tend to result from failure to adequately account for security-related concerns such as error handling, policy enforcement, and component composition. The total absence of security being taken into consideration is also a possibility (Tsipenyuk, Chess, & McGraw, 2005).

3. **Implementation phase**

   Vulnerabilities are introduced through the use of type-unsafe languages, insecure application programming interfaces (APIs), improper use of secure APIs, seeding by malicious developers, lack of secure coding practices, and/or simply developers not being educated on which defects are traditionally exploited by attackers (Tsipenyuk et al., 2005).

4. **Verification/testing phase**

   Vulnerabilities in this phase are a result of failing to catch vulnerabilities introduced in the requirements, design, and implementation phases. Moreover testing personnel may lack the knowledge and skill needed to probe for security weaknesses. The testing techniques employed may not take into account the software architecture and so fail to uncover pertinent issues (McGraw, 2004).

Generally speaking, the earlier in the SDLC the vulnerability is introduced, the more difficult and costly the corresponding defect would be to patch. For example, it might be fairly easy to patch a one-line coding error whereas a "patch" in the traditional sense would be near impossible to "apply" to a flaw rooted deep within the design. In such a case, a complete re-design might be required to address the design flaw (Hein & Saiedian, 2009). Thus, it is important to develop software using models that take security into account right from the requirements phase.

2.3.2 Security Considerations in the Software Development Lifecycle

Even though there is no guarantee that an organization can build 100% bug-free software, there are higher chances of them building good quality secure software by following a solid design process that emphasis on good design practices such as code review, risk management etc. (Noopur, 2005).

Developing secure products is a complicated and challenging task, however, by integrating security from the beginning of the development cycle, these challenges can be better addressed. Table 2.1 shows the considerations necessary for security to be incorporated throughout the software development lifecycle (Gupta et al., 2007).

Phase	Key Security Consideration
Requirements	• Determine security assumptions • Identify critical assets to be protected • Outline 3rd party security requirements and interface specifications. • Outline the requirements for securing the data storage, communication, and product configuration. • Carry out threat analysis from a high-level perspective. • Decide on applicable product hardening techniques.
Design	• Define the security architecture that will host the product. • Perform detailed threat analysis on sensitive assets. • Follow principles of defense in depth. • Use static and dynamic analysis tools.
Implementation	• Secure the development environment. • Follow security standards and best practices. • Perform code review and make use of security tools.
Testing	• Confirm that all security mechanisms are working as intended. • Analyze any defects found to prevent future occurrence. • Perform stress tests to identify vulnerabilities.

Table 2.1 Security considerations in the software development lifecycle.

Over time, many secure SDLC models have been proposed. A survey of existing security oriented processes, models, and standards conducted by (Noopur, 2005) discovered that they tend to focus in any of the following four areas: Security engineering activities, security assurance activities, security organizational and project management activities and security risk identification and management activities. Security engineering activities consist of activities such as security requirements elicitation, secure design, and static analysis tools etc. which are needed design a secure system. Examples of security

assurance activities are evaluations, validations, and verification. Organizational activities relate to the organization and include, policies, support from decision makers, roles etc. Project management activities are those activities that support secure design such as planning and tracking. Security risk activities are amongst the most important and they determine the rest of the activities. Examples of models include Capability Maturity Models such as the Trusted Model, Federal Aviation Model, and Systems Security Model. Additional models are Team Software Process for Secure Development, Correctness by Construction, Agile Methods, Secure Development Lifecycle and Common Criteria.

An example of a popular secure SDLC process is the Trustworthy Computing Security Development Lifecycle (SDL). This is a process that Microsoft uses to build secure software (Lipner, 2004). It consists of adding security focused steps to each of the software development lifecycles. During the requirements phase, two things happen. The development team is allocated a security advisor, and they also consider which security features will be integrated into the development process. The role of the security advisor is to advise the developers on what security milestones to aim for based on the size, complexity and risk of the project, user requirements as well as industry standards. Before selecting security features, the development team needs to consider how the security features of its software will interact with other software, how they will maximize security while minimizing any disruptions.

2.3.3 Vulnerability Management Systems

According to (Khan, 2011), there were approximately 157 vulnerability databases. Among those that are active, the major ones are shown in figure 2.2 below.

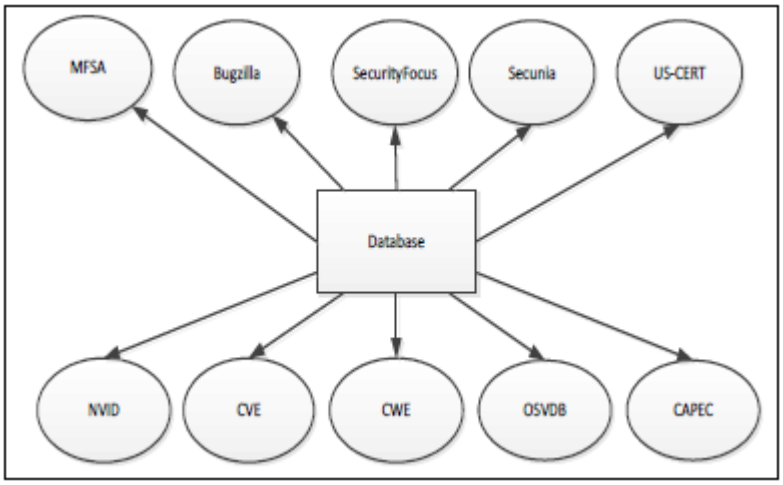

Figure 2.2 Major Vulnerability Databases source (Khan, 2011, 5)

The Mozilla Foundation Security Advisories (MFSA) is a vulnerability database maintained by the Mozilla Foundation ("Mozilla Foundation Security Advisories," 2016) . It contains a list of detected vulnerabilities and each record is classified based on a fix, product, reporter, time announced, its impact and a given title. It also contains links to relevant files, code, and patches via Bugzilla, bug tracking system.

The National Vulnerability Database (NVD) is maintained by United States government ("National Vulnerability Database," 2015). Vulnerability data is classified based on misconfigurations, impact metrics, security checklists, product names and security related software flaws.

The Metrics Database Program (MDP) is maintained by the National Aeronautics and Space Administration (NASA). It is a repository of data on errors in software and metrics at the function level. The identification of error records and affected product names is based on unique numeric identifiers. The metrics kept include: McCabe, Halstead, error, requirement, and Line of Code metrics (NASA, n.d).

The Microsoft Security Response Center (MSRC) is a repository maintained by the Microsoft Corporation ("Microsoft Security Response Center," 2016). It contains information on vulnerabilities found in Microsoft products.

The Common Vulnerabilities and Exposures (CVE) is a publically available database maintained by the MITRE corporation (MITRE Corporation, 2015a). It acts as a dictionary of all known vulnerabilities and exposures. Each record has a globally unique identifier known as the CVE ID.

The Common Weakness Enumeration (CWE) is a publically available database also maintained by MITRE (MITRE Corporation, 2015). It is a repository of information on weaknesses found in the code, implementation and software architecture. It also has security tools that can be used to detect weaknesses in the source code.

The Open Source Vulnerability Database (OSDVB) is a publically available database maintained by the company RiskBased Security. It contains information on security vulnerabilities found in computerized equipment ("Open Source Vulnerability Database," 2013).

The Common Pattern Enumeration and Classification (CAPEC) is a publically available database maintained by MITRE. It is a repository provides a standard way of identifying, collecting, refining and sharing of attack patterns ("Common Attack Pattern Enumeration and Classification," 2016).

Bugzilla is a publically available database maintained by the Mozilla Foundation. It provides a mechanism for tracking vulnerabilities in software, submission, and review of patches, team communication and management of quality assurance ("Bugzilla," n.d.).

SecurityFocus is a publically available database. It uses a mailing list platform called BugTraq to communicate vulnerability information. Each vulnerability is classified using a CVE identifier and a BugTraq identifier ("SecurityFocus," n.d.).

Secunia is a Danish public database maintained by Flexera Software ("Secunia," n.d.).It collects vulnerability information related to all known operating systems and a large number of software systems.

The r United States Computer Emergency Readiness Team and is a publically available repository maintained by the United States Department of Homeland Security ("US-CERT | United States Computer Emergency Readiness Team," n.d.). Vulnerability information is stored in the form of notes which are tagged using vulnerability note identifiers.

2.4 Crowd-sourced Testing Systems

Crowd-sourced software engineering involves placing an open call to software engineers to undertake various tasks in software engineering such as coding, requirements

elicitation, design, and testing. (Mao et al., 2015) The general process and actors in a typical crowd-sourced software engineering engagement are shown in figure 2.3.

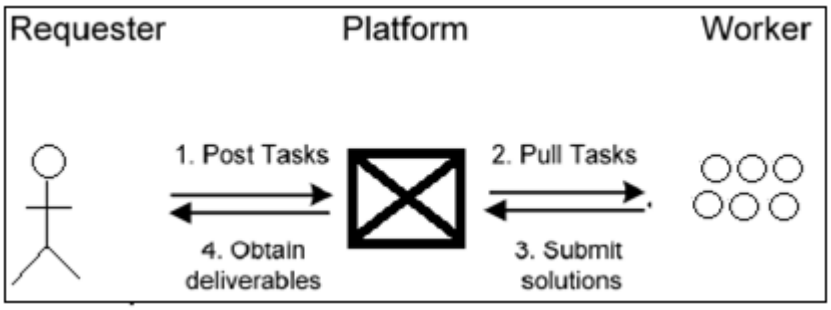

Figure 2.3 Actors and processes in Crowd-sourced Software Engineering

Testing in software engineering can also be crowd-sourced. This is usually abbreviated to crowd-testing. The workers consist of a team of software testing experts who are spread out across different geographies. They have different types of devices on which to carry out their tests. These devices run a multitude of operating system versions, web browsers, applications and human languages. These allow testers to carry out tests in different scenarios under different kinds of loads (Mellacheru & Swaminathan, 2014).

2.4.1 Review of Existing Crowd-testing Systems

This section reviews existing crowd-testing systems. In a case study, Zogaj, Bretschneider, & Leimeister (2014) outline the working principal of a commercial crowd-testing system. The process is as follows. Once a testing company receives a testing project from a customer, they start by determining the customer's testing requirements. This involves outlining the appointed devices, operating systems, and browsers on which the testing will be conducted. Next, the scope of testing is determined in terms of the duration of testing and the number of testers required. The outcome of this two processes is a framework that will guide the testing.

In the next stage in the process involves the testing company making an announcement on its testing platform targeting all testers. They then upload the application to be tested onto their platform and make it available to testers. The testers then review the application by looking for bugs. Once a tester detects a vulnerability, they record it and then submit a report. These reports are then evaluated by the project manager and the customer to discover which bugs to take into consideration. A report with the

registered bugs is then sent to the customer's development team. This report can then be imported into an issue tracking system such as Bugzilla or JIRA (Zogaj et al., 2014).

Customers have full visibility into the testing process and can intervene so as to alter the testing requirements and the scope. They also have the option to get different types of tests for applications in different stages of development. Testing is available on demand even after working hours or on weekends. Customers and testers each have personal profiles. On their profiles, customers can set up a testing project such as indicating specific requirements, device testing type, the scale of testing and testing procedure. On their profiles, testers have access to a dashboard which shows them their bug statistics as well as identified bugs that they have not yet been paid for. Figure 2.4 below outlines this process diagrammatically (Zogaj et al., 2014).

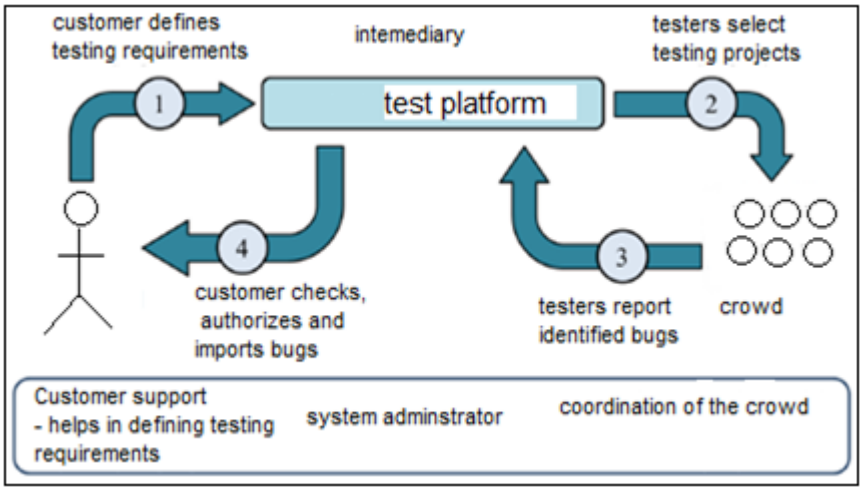

Figure 2.4 Crowd-testing process and workflow. Zogaj et al. (2014), 7

2.4.2 Bugcrowd

Bugcrowd was founded in January 2013 ("Bugcrowd," 2016). It is a San Francisco-based security company that provides crowd-sourced security solutions. It brings together security testers from around the world to work on discovering security bugs in both mobile and web applications. It uses a bug bounty program where hackers gave financial compensation for bugs they discover. The company provides its clients with reports highlighting bugs that have been discovered.

Bugcrowd uses a platform called Crowdcontrol for engaging its researchers, management of vulnerabilities and payout of rewards to researchers. Its key strengths include the fact that it offers researchers an environment to conduct sandboxed testing. Through the use of virtual private networking, researchers can view specific program content through their portal. This gives the ability to limit what researchers can or cannot access. They also offer flexible rewards which are shorter in duration than traditional penetration tests. This gives its customers the ability to control the scope and related costs. Bugcrowd operates a managed platform which means that reports presented by the crowd sourced testers will be reviewed by Bugcrowd analysts ("Bugcrowd," 2016).

2.4.3 Hackerone

Hackerone is a San Francisco-based company founded in 2012 ("HackerOne," 2016). That runs a vulnerability disclosure program which connects security researchers with businesses. Through their platform clients can opt to run a vulnerability coordination program for free as a private program. They also have access to the following features: a security page where they can obtain reports, gain access to a global community of security researchers with vetted profiles and reputation scores, a workflow to manage reports as well as integration into an issue tracker of choice.

Hackerone's has several strengths. They have a feature called Hackbot machine learning which offers automated de-duplication of reports. Bugs can also be automatically be associated with publically known reports to facilitate closure. In addition to providing a hosting platform to manage the testing program, they can also train staff on how to run a crowd-testing program. They also facilitate collaboration with third party consultants for organizations that lack human resources. Additionally, they can use an open source model, unlike other testing platforms ("HackerOne," 2016).

2.4.4 Synack

Synack was launched in May 2013 in Redwood City, California ("Synack," 2016). It offers its clients a safe way to crowdsource vulnerability intelligence. It uses a

network of freelance security analysts to test the networks of customers who have signed up for its subscription-based services. Synack researchers make use of a platform called Hydra that aids researchers by looking not only for security bugs but also out of date software and other concerns. If Hydra finds something, it sends an alert to a researcher who can then do further investigation. For mobile, Hydra also takes into account issues such as insecure cryptography, embedded keys, and passwords etc.

Synack's key strengths include: For organizations with strict requirements on researcher identities, Synack offers background checking of a private crowd of testers. They can also bring in experts to provide a debriefing on critical security vulnerabilities. In terms of costing, they can run a program for items in scope at a flat rate which allows clients to easily budget the costs of running the program. The use of its platform Hydra, helps analysts to quickly focus their attention on critical areas instead of relying on their own methodologies which can be time-consuming. Hydra is also not solely focused on tracing security bugs, but also alerts on out-of-date software ("Synack," 2016).

2.4.5 Limitations of the Reviewed Systems

From a general point of view, all the testing platforms suffer similar limitations which are described as follows. It is important to note that the strengths of one platform can be considered as the limitations of the other. To begin with, these platforms generally do not offer a service to fix vulnerabilities discovered. The fix is left to the developer requesting the service. They also do not test whether a particular application may be malware. The expectation is that the customer will use the reports from testers to fix their own vulnerabilities. This can be time-consuming for customers. Some projects require that the requestors withhold information for the sake of confidentiality. This can create issues with the crowd not fully understanding the task required of them due to missing information (Steinhauser, 2013).

Most of the challenges relate to how the various platforms manage the crowd of testers. Some tasks require more highly skilled labor than others. This can act as a barrier that limits the number of potential testers on a project. Furthermore, some tasks are quite

complex and need to be broken down into smaller tasks by the project manager. This might prove to be a challenge for certain types of projects (Zogaj et al., 2014).

Real-time communication is another challenge. On these platforms results are rewarded on a first come first served basis, thus, there is the potential of having duplicate work (Zogaj et al., 2014). Testers in the crowd lack the ability to interact with each other in real-time during the testing process. The fact that testers cannot clarify on instructions from developers means the quality of work delivered is of poor quality (Zhang, Chen, Fang, & Liu, 2016).

There is a lack of real-time communication between testers and developers. This may result in several testers reporting the same vulnerability leading to wasted time and effort. Testers also lack a way of communicating amongst themselves during projects to facilitate knowledge transfer (Zogaj et al., 2014).

The heterogeneous nature of the crowd also makes guaranteeing quality as well as data security a challenge. Since the testing crowd consists of people from different backgrounds and skill levels with varied timetables, careful planning and scheduling are required. This can create a lot of overhead for the project manager as he/she tries to make sure only the best resources are assigned a task. It is also a challenge to estimate the how many testers it would take to deliver a certain project within a certain time frame. There is also the challenge of ensuring that testers with a particular set of skills will be motivated to participate in a certain testing project (Zogaj & Bretschneider, 2014) .

2.5 Conceptual Framework

This dissertation is based on the framework shown in figure 2.5

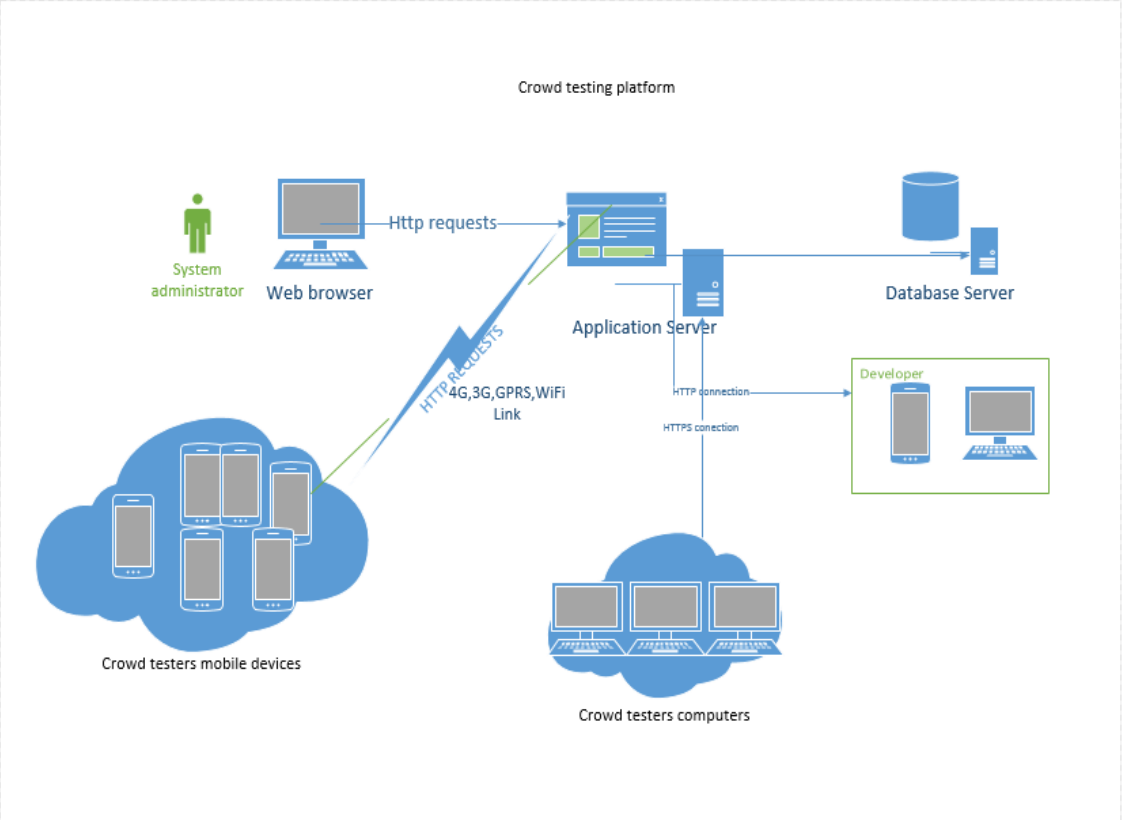

Figure 2.5 Mobile-based crowd-testing system

The system allows access to the developers, testers, and the system administrator through their computers as well as via an application running on their mobile phones. The system consists of an application server that hosts a web application, the testing platform and it connects to a database server. Users in the system can transmit information from their mobile phones to the servers as well as to other users. Communication between the web client and web server is HTTP based.

The mobile application, which this study intends to design, will provide a way for testers to receive real-time updates about new testing projects. It will also enable them to submit summary reports of their tests as well as send messages to other users on the system. The developers use their Internet browsers to access the web server. From here they can send summarized project information to crowd testers. The testers then receive an alert that a new project has been created and they can look at it after connecting to the system with their mobile applications. Developers and the System administrator can also access submitted reports as well as receive messages via the mobile application.

2.6 Conclusion

This chapter has reviewed methodologies used in developing software and the way in which they can be improved to reduce the number of vulnerabilities found in software. It provided insight into how crowd-sourced software testing works and how developers and testers make use of them. Existing crowd-testing systems were reviewed and gaps in them identified. From the literature review, one of the gaps found in these systems is that they generally lack a way of facilitating real-time communication between users in the system. This relates to how projects are announced or reports submitted and the need for consultations during testing. This dissertation proposes to design and develop a system that will address these challenges.

CHAPTER 3 RESEARCH METHODOLOGY

3.1 Introduction

This chapter will describe how the research was designed, participants selected and research instruments used in an attempt to address the research questions raised in chapter 1. Furthermore, this study will use evolutionary prototyping during the pilot phase. The justification for using this method is because prototypes help to reduce the time and cost required to build the overall system. If there happens to have been a mistake in the early phases of the SDLC, a prototype can help address this before it gets propagated through to later stages of the SDLC. (Carter, Anton, & Williams, 2001).

3.2 Research Design

This research in this dissertation will be conducted using an exploratory design. This type of design is useful in gaining an understanding of the best methodology to apply when gathering information (Labaree, 2016). This approach is flexible and can address all types of research questions, however, it generally utilizes a small sample size and so findings cannot be generalized to the whole population.

Qualitative research methods were used as the method of obtaining data. This method was chosen because it allowed the researcher to obtain information on how the users perceived the system. This type of information would be useful in the further development of the system to a point that it would effectively meet the user's needs.

3.3 Participants

In research, when the number of elements in a population are unknown or cannot be individually identified, a non-probability sampling design is the recommended approach. The selection of elements will depend on other considerations. Purposive sampling relies on the judgment of the researcher as to who can provide the best information that will meet the desired objectives of the study (Kumar, 2011).

In this study purposive sampling design was used. The study aims at building a universal information sharing tool. The worldwide crowd-sourced tester population is not known. The target population consisted of software developers and testers who had

experience developing software for use in the Kenyan financial industry. They were also chosen because they are involved in the day to day development and testing of such software. 8 participants were contacted and 6 agreed to participate in the study.

3.4 Research Instruments

The main data gathering instrument used was a questionnaire (see Appendix A). It had four main sections: a personal profile, a general section, a features section and a user experience section. The profile sought to collect some socio-demographic characteristics of the respondents. The general section sought to collect data on the users experience with mobile phones and crowd-soured software testing platforms. The features section asked the users to rate the mobile application's features and lastly the user experience section sought to find users perception of the application.

The questions were structured using the Likert format. Five choices were provided for every question or statement. The choice made represented the degree of agreement each respondent had on the given question. Thereafter, a weight was calculated and used to interpret the total responses.

3.5 Pilot

This research made use of the Evolutionary prototyping methodology to design the system. This is because it allows for quick simulation of the known features of the eventual system while at the same time allowing for changes to be made as new requirements are discovered.

The Evolutionary Prototyping process has six phases, namely: Analysis of requirements, Design of the system, Implementation/Building of the prototype, System testing, User testing, and Delivery to end client. In practice, the phases often overlap and iterate severally until the final system is built. The following paragraphs describe how each phase of the evolutionary prototyping cycle was referenced in this study.

3.5.1 Requirements Analysis

According to Lutz and Woodhouse (1997), a requirement describes what the system should do. The process involves identifying stakeholders as well as the analysis, documentation, and validation of requirements.

In this study, stakeholders were identified based on the researchers experience with working in the field of software development security. Additionally, literature was reviewed and presented in Chapter 2 of this study. This was useful in establishing how existing crowd testing systems worked as well any existing gaps. Key features in the existing crowd testing systems were identified for incorporation in the prototype that will be developed. Use-case diagrams and use-case descriptions were used to document the requirements (Jacobson, Jonsson, & Overgaard, 1992).

3.5.2 System Design

The physical architecture of the system was outlined with the aid of a network diagram. The main classes, objects, and interactions in the system were visually represented using components derived from the Unified Modeling Language. These include use case diagrams, class diagrams, sequence diagrams. Additionally, data flow diagrams, entity diagrams, and the database schema were also used.

3.5.3 Implementation

The prototype was developed based on the design documentation. It was modified in an iterative fashion as new requirements were received from the users. The client side of the system consisting of a mobile application was implemented on the android platform. The server side consisting of a web application was developed using PHP and MySQL programming languages. The server system runs on an Apache web server.

3.5.4 System Testing

At this stage, the researcher and potential users were exposed to the prototype to judge whether the prototype met their desires. Where the results were found unsatisfactory, changes to the requirements, design and prototypes were carried out. For those results that were satisfactory, new iterations for a new requirement were done.

3.5.5 Software Evaluation Survey

This goal of this stage is to determine whether the features in the prototype are sufficient to make it a suitable information sharing tool for use by crowd testers. Respondents chosen were shown an early version of the mobile application and given a questionnaire to fill.

The first part of the questionnaire assessed the respondents experience with working with mobile-based information sharing tools on crowd-sourced testing systems. Later they were asked to rate the features of the prototype on a Likert scale.

3.6 Research Ethics

To ensure adherence to ethical codes of conduct, informed consent was obtained from respondents. They were informed that all information provided would be confidential and the purpose of the study was explained to them. They were also given the option to withdraw from the study at any time they wished.

3.7 Conclusion

This chapter has discussed the methods that would be used to collect data using qualitative methods and make sure the research met the intended objectives and answered the research questions.

CHAPTER 4 SYSTEM ARCHITECTURE AND DESIGN

4.1 Introduction

This chapter describes the architecture and design of the mobile-based crowd testing system that satisfies the requirements discovered during the prototyping phase.

.

4.2 System Architecture

The system architecture of the proposed system can be represented in three tiers. These are the presentation/user layer, the application layer, and the data layer. The presentation tier presents data to the user. In figure 4.1, the mobile application and the web browser client from this layer. The application tier performs the application logic. It provides enables clients in the presentation layer to access the data layer. It consists of the web server, web scripting language. In the proposed system the web server is an Apache server and the scripting language is PHP. The data layer consists of a database management system. This is a MySQL database in the proposed system. It provides storage and retrieval of data within the system.

4.2.1 Network Diagram

The system consists of three main components shown in figure 4.1. These are a mobile application, a web application, and a database.

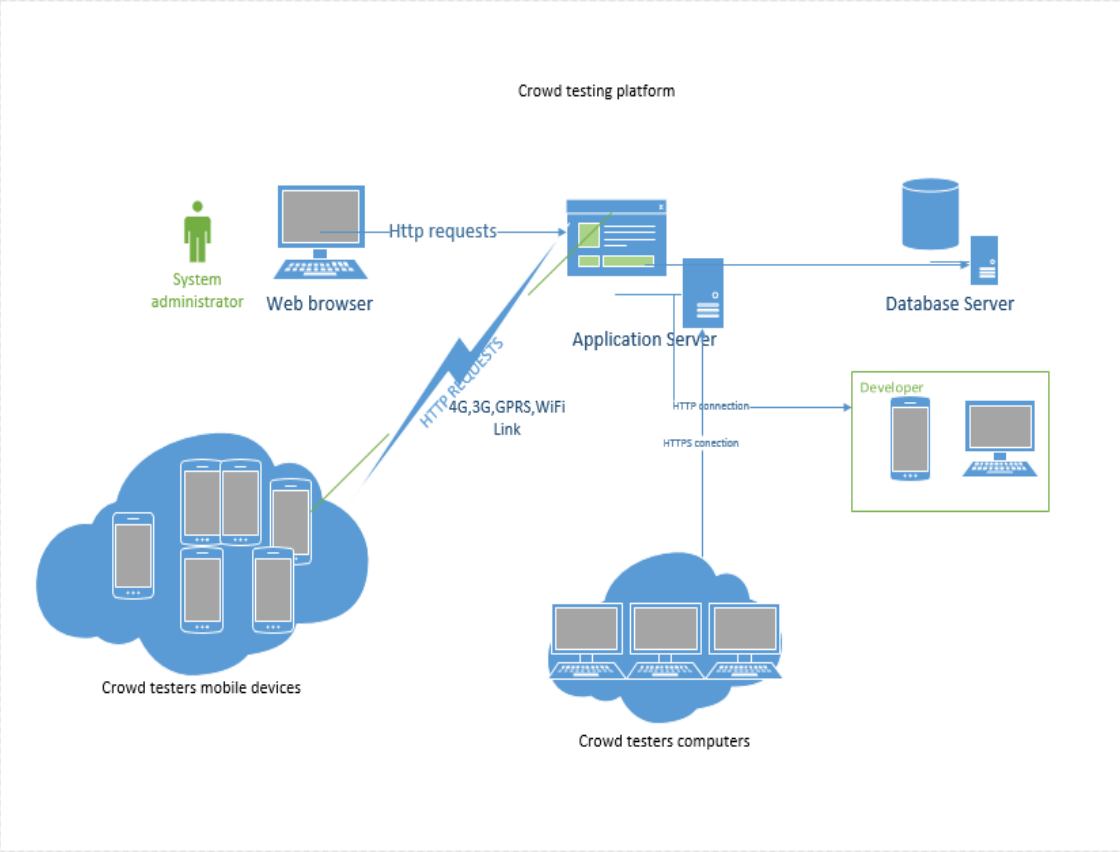

Figure 4.1 System architecture

The mobile-based and browser based clients communicate with the application server via HTTP requests. The information flow is bidirectional. All requests to the database first pass through the application server. As shown in figure 4.1, testers use the mobile application running on their mobile phones to communicate with the application server. The system administrator and developer can access the system via a website. Access to the system is regulated by the use of user credentials.

4.2.2 Databases

The application consists of two subsystems each having a database. The mobile application is able to store data to a local database in the phone memory as well as to the external database hosted on the web server. Data stored on the web server comes from the message module, reporting module, projects module and the user module. The local storage stores preference information for the mobile application.

This server runs on the MySQL platform and its key purpose is to store tables that contain data consisting of reports, messages, projects and user information.

4.2.3 Security

The data transmitted between the mobile application and the backend uses secure channels and avoids open networks. Furthermore, users are required to log into the systems using correct credentials.

4.2.4 Mobile Device

The client running on the mobile device is an android based application. It consists of four main modules. The projects module allows users to access projects that they are currently working on. The reporting module enables testers to send reports about bugs they have found. The messaging module facilitates communication between users found in the system. The profile module allows testers to update their user profiles.

4.2.5 Application Server

This consists of an Apache web server and the server that supports the crowd-testers. The web server serves the pages of the web application to the mobile and web clients. The web application is PHP based and has the following modules: projects, reporting, messaging and user modules. Administrators access it via the browser client and can perform tasks such as allocating new projects, receiving reports and managing system users. The crowd-testing platform is beyond the scope of this study.

The web client consists of an interface powered by HTML, JavaScript, and Cascading Style Sheets (CSS) programming languages. It enables users to access all the modules provided by the web application server.

4.3 System Design

This section provides a detailed analysis of the system architecture by making use of Unified Modelling Language (UML) diagrams as well as narrations. The UML diagram consists of Use case diagrams, Data Flow Diagrams, System Sequence Diagrams, Class diagrams and the Entity Relationship Diagrams.

4.3.1 Use Case Modelling

The top level use case diagram of the system is shown in figure 4.2. The main actors are the developers who create the applications to be tested, the testers who evaluate the applications to be tested and the system administrators who manage users and communication within the system.

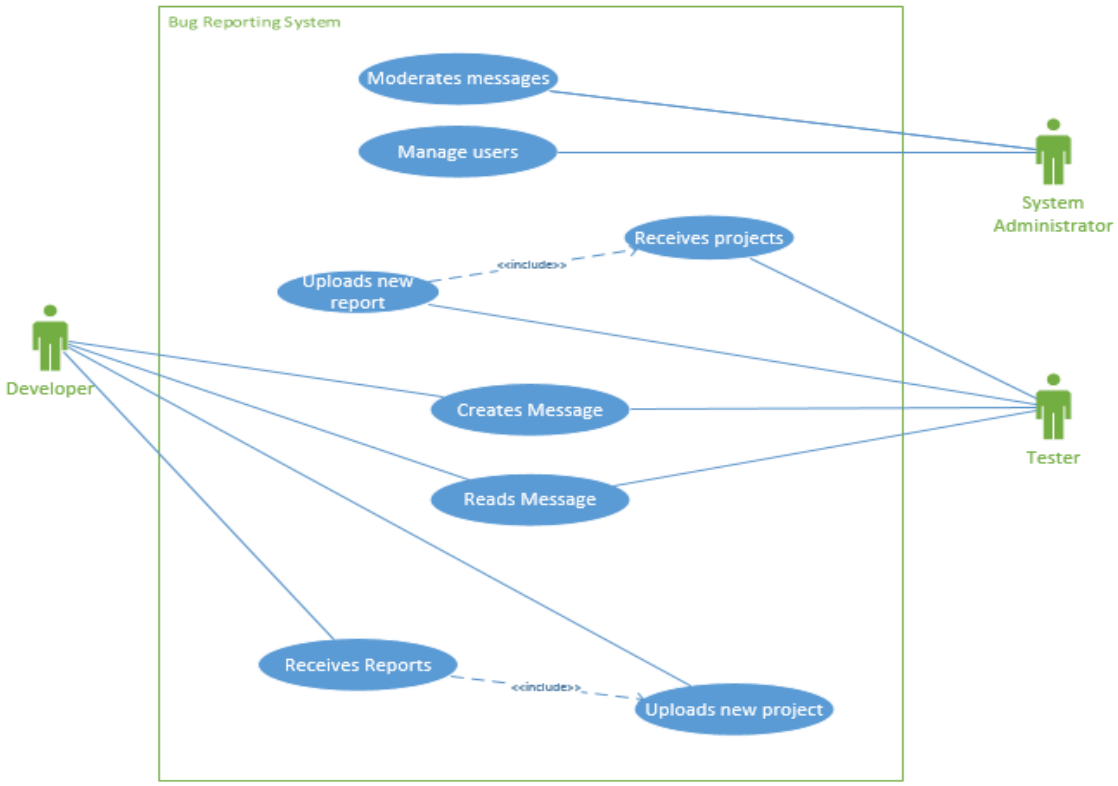

Figure 4.2 Use Case Diagram for Information Sharing System

Use Case 1: Upload a Vulnerability Testing Report

This use case outlines how a tester can submit a report on a vulnerability or bug they discover in an application they are testing. This use case is extended by the "receive new project" use case.

Use Case Name:	Upload new Report
Scenario	Done via the mobile application

Trigger	Tester has completed testing	
Brief description	Tester reports a vulnerability they have discovered by adding details such as the name of vulnerability, description, date and severity.	
Actors	Tester	
Stakeholders	Tester, Developer	
Preconditions	Tester has logged into the application	
Post-conditions	A new report is created	
Flow of Events (steps)	**Actor**	**System**
	1.Tester inputs report details and clicks submit	1.1 System stores a new report in the database.

Table 4.1 Submitting a Vulnerability Report use case

Use Case 2: Retrieve a Vulnerability Incident Report for Modification

This use case outlines how a user can retrieve and update a report on a vulnerability incident they had previously created. This use case is extended by the upload new project use case. A report must exist to be viewed.

Use Case Name:	Retrieve a Vulnerability Incident Report	
Scenario	Done via the mobile application or web application	
Trigger	The user wants to view or make an update to the report	
Brief description	The user uses the mobile or web application to submit a report ID for a report they had created in the past and they are presented with the report.	
Actors	Tester, Developer, System Administrator	
Stakeholders	Tester, System Administrator, Developer	
Preconditions	The user has logged into the application and the desired report had been previously created	
Post-conditions	The report is retrieved	
Flow of Events (steps)	**Actor**	**System**

	1.the user inputs report ID and clicks submit	1.1 System retrieves the report based on its ID.
	2. The user views, adds or deletes details to the report and clicks Save	2.1 System associates the report to the previous report.

Table 4.2 Retrieve a Vulnerability Report use case

Use Case 3 Receive a New Project

This use case outlines how a user can retrieve a new project to test for vulnerabilities.

Use Case Name:	Receive Project	
Scenario	Done on both mobile and web application	
Trigger	Tester wants to select a project	
Brief description	Tester retrieves project information using the mobile or web application	
Actors	Tester	
Stakeholders	Tester, Developer	
Preconditions	Tester has logged into the application	
Post-conditions	The project is selected.	
Flow of Events (steps)	**Actor**	**System**
	1.Tester opens the projects available module	1.1 System presents projects available
	2. Tester selects the project they want to work on.	2.1 System associates the selected project with the tester.

Table 4.3 Retrieve project use case

Use Case 4 Read Message

This use case outlines how a user can read messages shared on the platform by other users of the system.

Use Case Name:	Read Message	
Scenario	Done on either mobile application or web application front end	
Trigger	The user receives notification a message has arrived	
Brief description	The user reads the message on their mobile application or web application.	
Actors	Tester, Developer, System Administrator	
Stakeholders	Tester, Developer, System Administrator	
Preconditions	The user has logged into the mobile or web application	
Post-conditions	The message is read	
Flow of Events (steps)	**Actor**	**System**
	1. The user opens the messages module and clicks the refresh icon	1.1 System populates the screen with the latest messages.
	2. The user reads the messages on the screen.	2.1 System updates the screen with the latest messages

Table 4.4 Read Message use case

Use Case 5: Compose Message

This use case outlines how a user can compose messages to be shared on the platform to other users in the system.

Use Case Name:	Create Message
Scenario	Done on either mobile application or web application front end
Trigger	The user desires to compose a new message
Brief description	The user composes a new message on either the mobile application or web application.
Actors	Tester, Developer, System Administrator
Stakeholders	Tester, Developer, System Administrator

Preconditions	The user has logged into the mobile or web application	
Post-conditions	The message is sent	
Flow of Events (steps)	**Actor**	**System**
	1. The user opens the messages module and clicks the new message link	1.1 System presents the input message screen.
	2. The user composes the message and presses send	2.1 System broadcasts the message to all the users as well as updating the messages dashboard.

Table 4.5 Compose Message use case

Use Case 6: Uploads a New Vulnerability Testing Project

This use case outlines how a developer can upload a new project for testing of an application's vulnerabilities.

Use Case Name:	Upload new Project	
Scenario	Done via the web application	
Trigger	Developer has an application they want to be tested	
Brief description	The developer creates a new project by uploading details such as the project title and description, project id, due date, scope of work, type of tests and any other relevant information.	
Actors	Developer	
Stakeholders	Tester, Developer	
Preconditions	Developer has logged into the application	
Post-conditions	A new project is created	
Flow of Events (steps)	**Actor**	**System**
	1. Developer inputs report details and clicks submit	1.1 System creates a new report.

Table 4.6 Submitting a Vulnerability Report use case

Use Case 7: Moderate Messages

This use case outlines how a user can access submitted projects and review them for approval.

Use Case Name:	Approves submitted messages	
Scenario	Done via the web application	
Trigger	The user submits a message on the platform	
Brief description	The user accesses the mobile application and sends a new message. The administrator sees the message and can decide to delete it if it violates company policy.	
Actors	Developer, Tester, System Administrator	
Stakeholders	Tester, Developer, System Administrator	
Preconditions	Tester and Administrator are logged into the system	
Post-conditions	A new message is created	
Flow of Events (steps)	**Actor**	**System**
	1.Administrator access messages module to view submitted messages	1.1 System distributes message.

Table 4.7 Approve Submitted Vulnerability Report use case

Use Case 8: Manage Users

This use case outlines how the system administrator manages users on the system.

Use Case Name:	Manage User
Scenario	Done via the web application
Trigger	The administrator receives a request to create a new user, delete user or update user details.
Brief description	The administrator accesses the user management web portal and performs the relevant information
Actors	System Administrator
Stakeholders	Testers, Developers, System Administrator

Preconditions	The administrator has logged into the application back end.	
Post-conditions	A new user was created, deleted or updated.	
Flow of Events (steps)	**Actor**	**System**
	1. Administrator access the user management module and creates, deletes or modifies user details.	1.1 System modifies user details accordingly

Table 4.8 Manage users use case

4.3.2 Data Flow Diagrams

The data flow diagram below shows how information flows between the various components making up the system. They consist of a context diagram, level zero, and level 1 diagrams

Context Diagram

This gives a bird's eye view of how information flows and the interrelationships amongst the main system components. The developers upload project details, send messages and receive messages and project information. Testers submit their profile details, vulnerability reports, and messages while at the same time receiving the message, and project information from the system. Administrators submit user information and messages while they receive messages. Figure 4.3 shows the context diagram that describes the system to be developed.

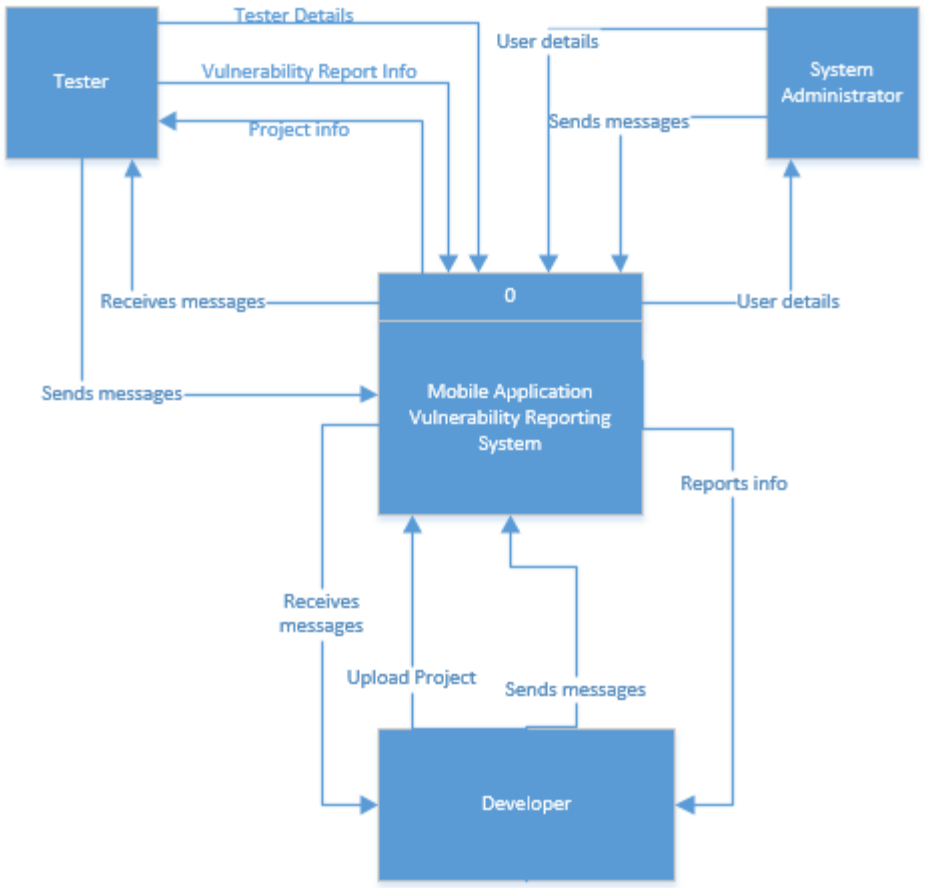

Figure 4.3 Context level diagram

Level 0 Data Flow Diagram

This is shown in figure 4.4. It provides a drill down of the system by showing more components and the way data flows between them. This is a granular look at each data flow presented in the context diagram. For instance to retrieve a report, developers have to provide the report details to the system which fetches it from the database using its unique identity number.

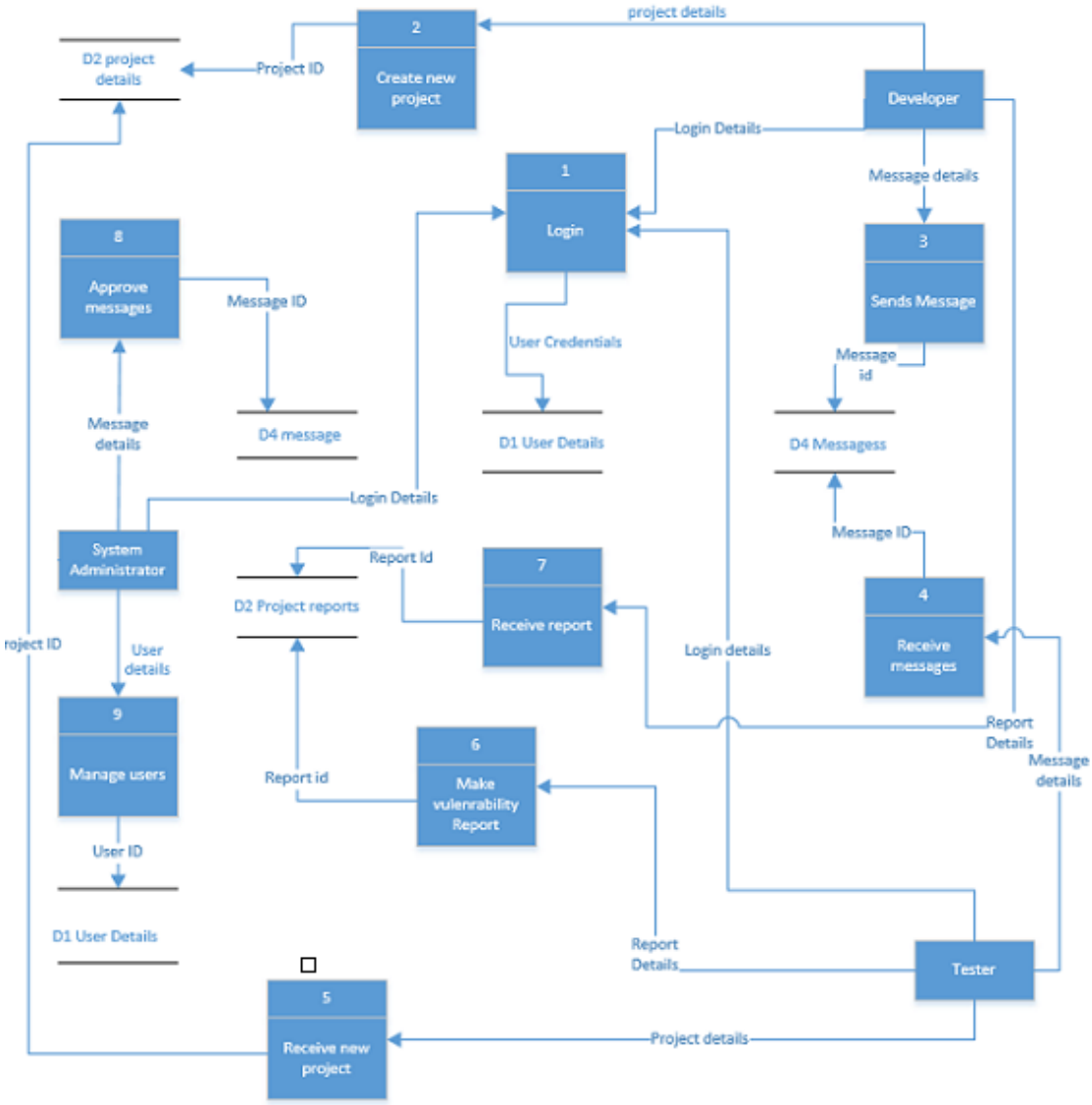

Figure 4.4. Level 0 Data Flow Diagram

Level 1 Data Flow Diagram for Making an Incident Report

This shows a drill-down of the data flow during the process of making a report by the tester. It is shown in figure 4.5.

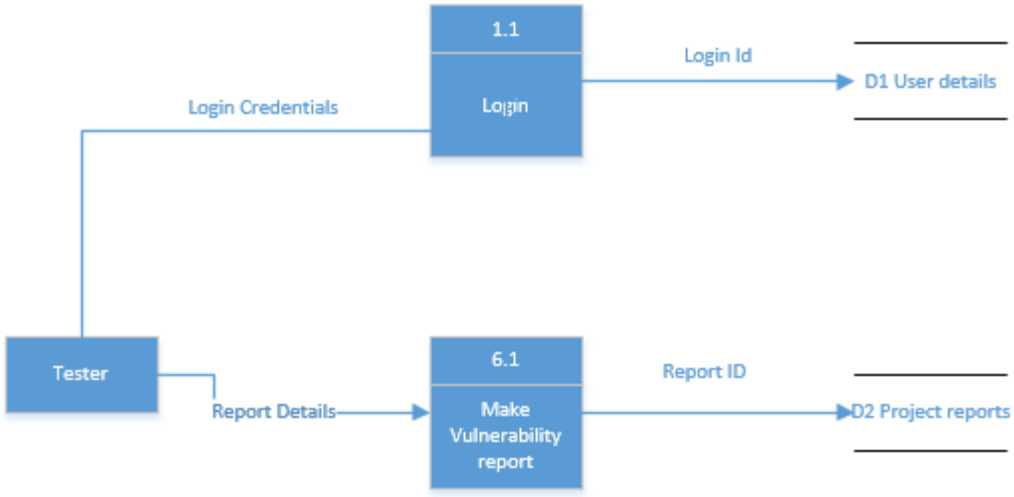

Figure 4.5 Level 1 Data Flow Diagram of Submission of a Vulnerability Report

4.3.3 Entity Relationship Diagram

This is a graphical depiction of the relationships between various entities in the system. The user entity represents the developers, testers, and administrators in the system. It has one unique primary key called the user ID. The rest of the properties such as first name, last name etc. are common to all. Each user, depending on their role can have more than one project, create more than one report and create multiple messages.

The reporting entity represents reports made in the system. Each report can only belong to one project and one user. The project entity represents the qualities of a typical project made in the system. Each project can only be created by one user, contains one or more reports and has multiple messages. The message entity represents messages made in the system. Each message can be created by only one user and it can exist across multiple projects. This is depicted in figure 4.6.

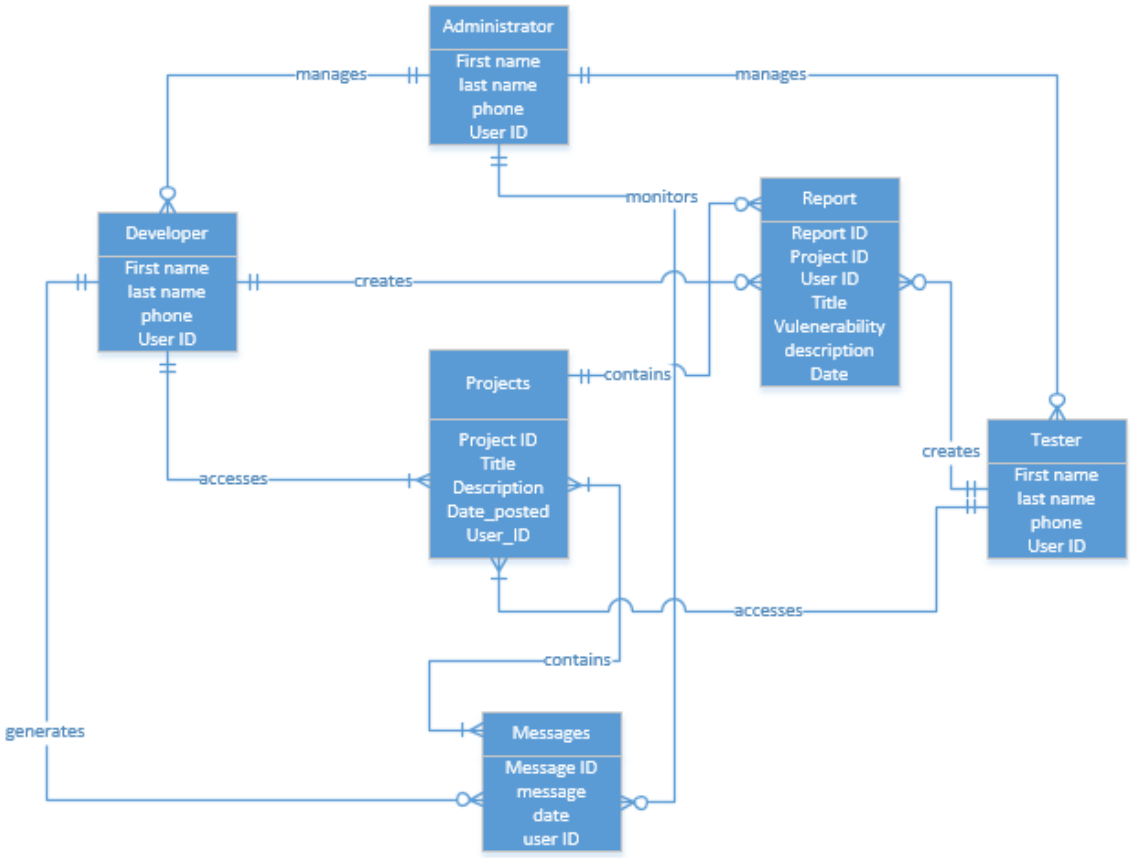

Figure 4.6 Entity Relationship Diagram

4.3.4 Sequence Diagram

This sequence diagram in figure 4.7 shows how information flows from the mobile application in the tester's phone to the web application running on the web server till it hits the database server. The tester can upload reports, receive new projects and send or receive messages.

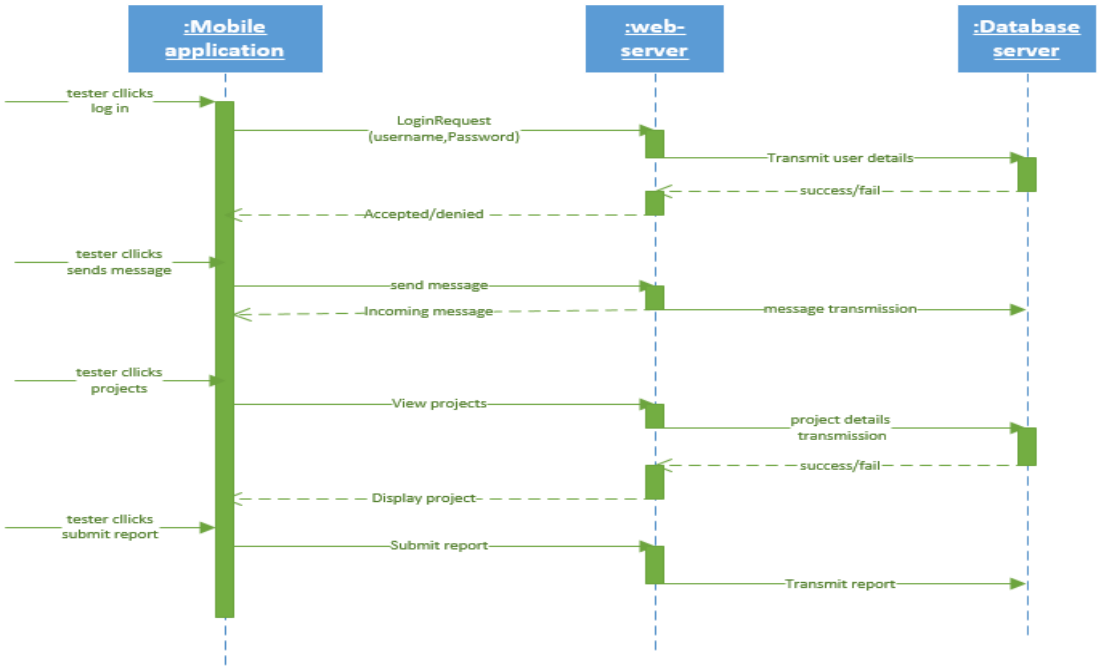

Figure 4.7 System sequence diagram

4.3.5 Class Diagrams

Figure 4.8 shows the various classes created in the system and the interactions between them. Each person in the system is linked to exactly one user. The user owns project, messages and report. The user has login credentials which serve as part of the unique id. The user could be in several states: new, active, blocked or banned. Project owns several messages. Every message, report, and project has a unique ID and belongs to exactly one user.

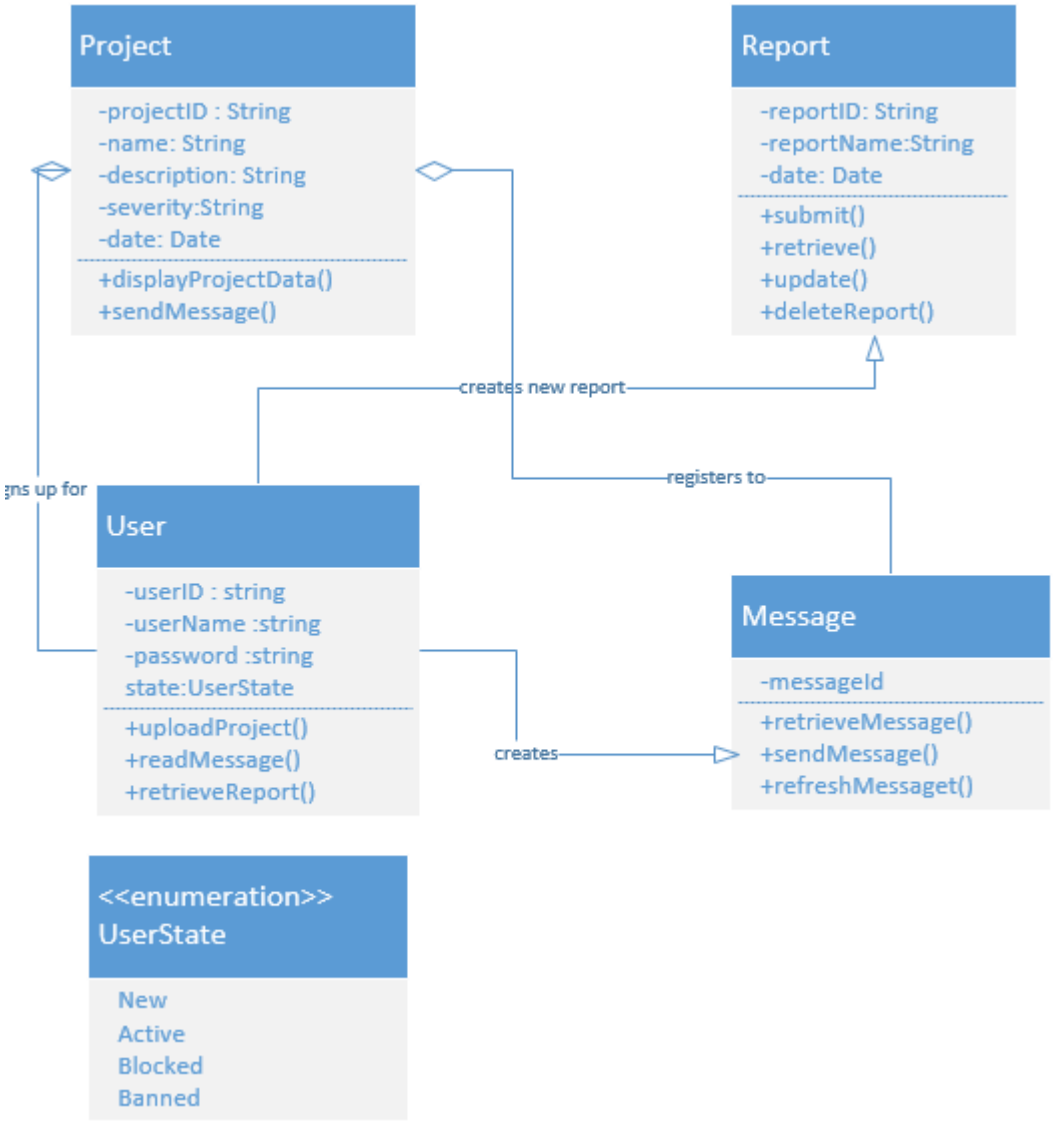

Figure 4.8 Class diagrams

4.3.6 Database Schema

The database design for the proposed systems is shown in figure 4.9. It has four tables which represent the main entities in the system which are users, reports, messages, and projects. It also shows the relationships between the various tables. Each table has a primary key which provides a unique identifier for the table and is set to auto increment and generates a unique number whenever a new record is inserted into the table.

The main entity in the system is the User entity. This tracks information about the various users in the system. All the other entities hold information that ties back to the

user table. The projects table stores information about new test projects. The message table stores information about messages shared in the system. The report table stores information about reports that testers have created.

One user can create multiple reports, messages and projects depending on whether they are a developer, a tester or a system administrator. One project can contain several reports. Whereas multiple messages can be contained in multiple projects.

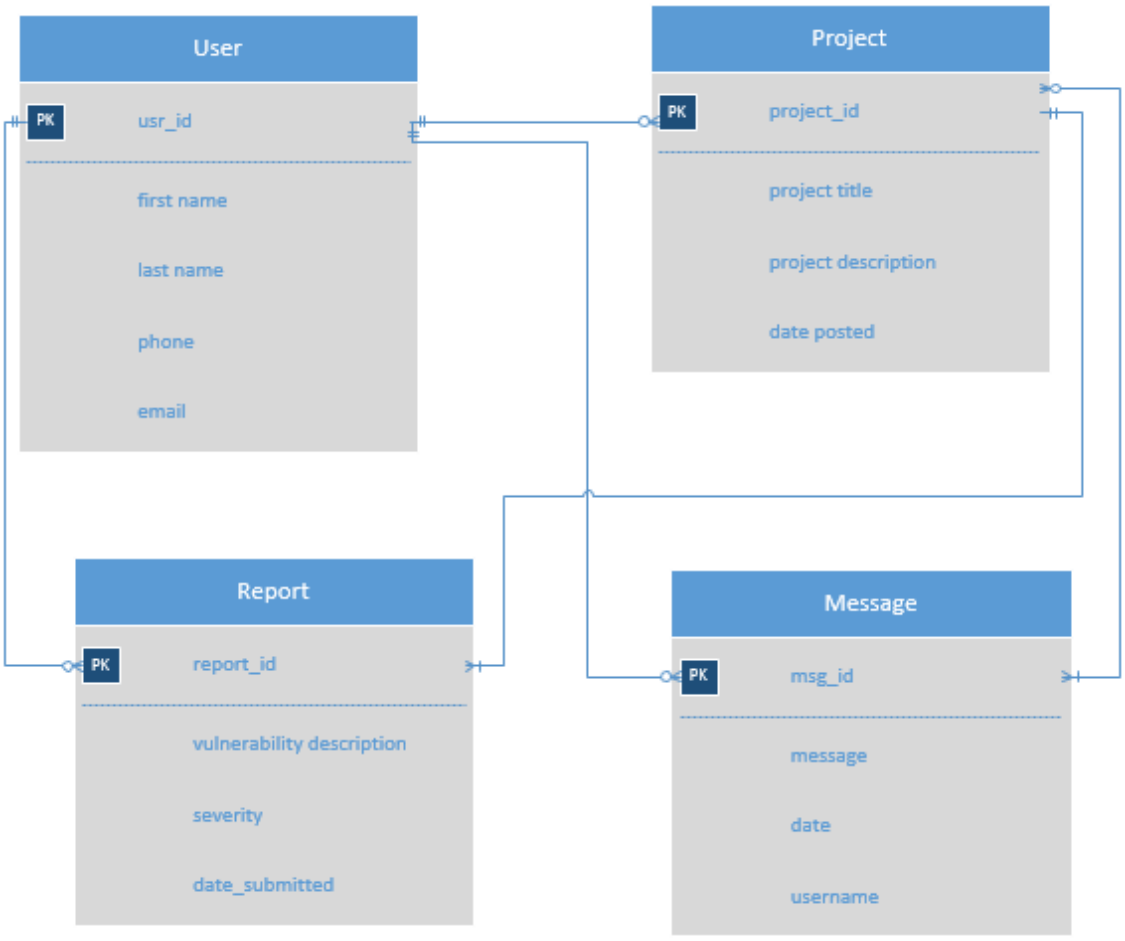

Figure 4.9 Database schema

CHAPTER 5 SYSTEM IMPLEMENTATION AND TESTING

5.1 Introduction

This chapter describes the implementation of the proposed system and highlights the key features. The features and functionality incorporated are the result of incorporating user views and requirements obtained from the evolutionary prototyping phase. This section also includes screenshots of the various end user and back end screens. The next major section of this chapter describes the various tests that were carried out on the system. It includes a discussion on the testing processes, locations, and a population that was taken into consideration.

5.1.1 System Specification

The web application runs on the following software:

- Windows 8 64-bit operating system
- Apache version 2.4.9 which is the web server
- PHP version 5.5.12
- MySQL version 5.6.17 which serves as the database

The web client was designed using PHP, JavaScript, HTML and CSS. The mobile application was designed in Java for Android. The database was administered via the tool PHPMyAdmin.

The web server and application server were installed on a computer with the following hardware specifications

- Hard disk size of 500 GB
- Memory of 8 GB RAM
- Intel Core i3 Central Processing Unit @2.10 GHz

5.1.2 System Implementation

Mobile Client Application

The application running on the mobile device was developed using the Java programming language and was designed to run on the Android operating system. Android is an operating system designed by Google for use in running smartphones.

Internet Brower Client Application

The browser client was developed using HTML, Ajax, CSS and JavaScript programming using the jQuery library. It is designed to run on any modern Internet browser such as Firefox, Chrome and Internet Explorer. The jQuery code is contained in a separate file and it allows for better user experiences as the code is executed on the browser instead of on the server. Ajax is also used to improve user experience by enabling the system to update parts of the web page without reloading the entire page. The CSS code is used to style the web page while HTML provides the basic structure of the web page.

Server side application

The web application was implemented using PHP (Hypertext Processor). PHP is an open source scripting language suitable for use in developing web applications. The PHP application is hosted on an Apache web server. Apache is an open source web server. The scripts provided a way for the client applications to connect to and pass data to and from the database.

Database Server

The database was implemented on a MySQL database server. MySQL is an open source Structured Query Language database that is distributed by the Oracle Corporation.

5.1.3 System Deployment

After the development of the mobile application, a .apk file was generated and installed into a Huawei Y300-100 smartphone running a 4.1.1 android operating system. The web server and database server were installed as one package on the Windows 8 computer mentioned in section 5.1.1. The created web application was then saved into the root folder of the web server. The server configuration web page was set to be accessible from the address www.localhost:8081. The web application was also accessible from this page.

5.2 System Features

This section is divided into three parts. General features found in both the mobile application and the web application, then specific features found in each and finally database specific features.

5.2.1 Creation and Publication of Projects

Projects are created using the web front end primarily by the development team. The project designer has a standard format that users have to follow when creating a new project. Access to the project details from the mobile device is dependent upon the mobile application user having logged into the system successfully. When a project is published it is available to all logged in users on the system. Refer to appendix B for screenshots of this page.

5.2.2 Creation and Publication of Reports

Reports are created using the mobile application primarily by the testing team. The report designer has a standard format that users have to follow when creating a new report. Access to the report from the mobile device is dependent on the mobile application user having logged into the system successfully. When a report is published it is available to all logged in users on the system. When a user submits a report the report details appear on their dashboard screen. Refer to appendix B for screenshots of this page.

5.2.3 Submission and Review of Messages

Messages are created using the mobile application by any user who is logged into the system. The users key in their message and then they press the send. The application connects to the web application running on the server and relays the message. The server can then resubmit the message to all users on the system in real-time.

User Management

All the users in the system are managed via the web back end. The administrator logs into the system and is able to create, update, delete or temporarily ban users. End users can also manage their own profiles from the mobile application on their phones

5.2.4 Mobile Application Features

The main components of the mobile application are the project view and retrieval, report submission, messaging, user profile management and security. The component diagram in figure 5.1 shows these modules.

5.2.4.1 Project Retrieval and Display

Projects are retrieved from the web application via HTTP URL calls. The URL contains the report identity number (ID) together with fields that make up a project. The mobile application then populates the phone screen with the project details. See figure 5.1.

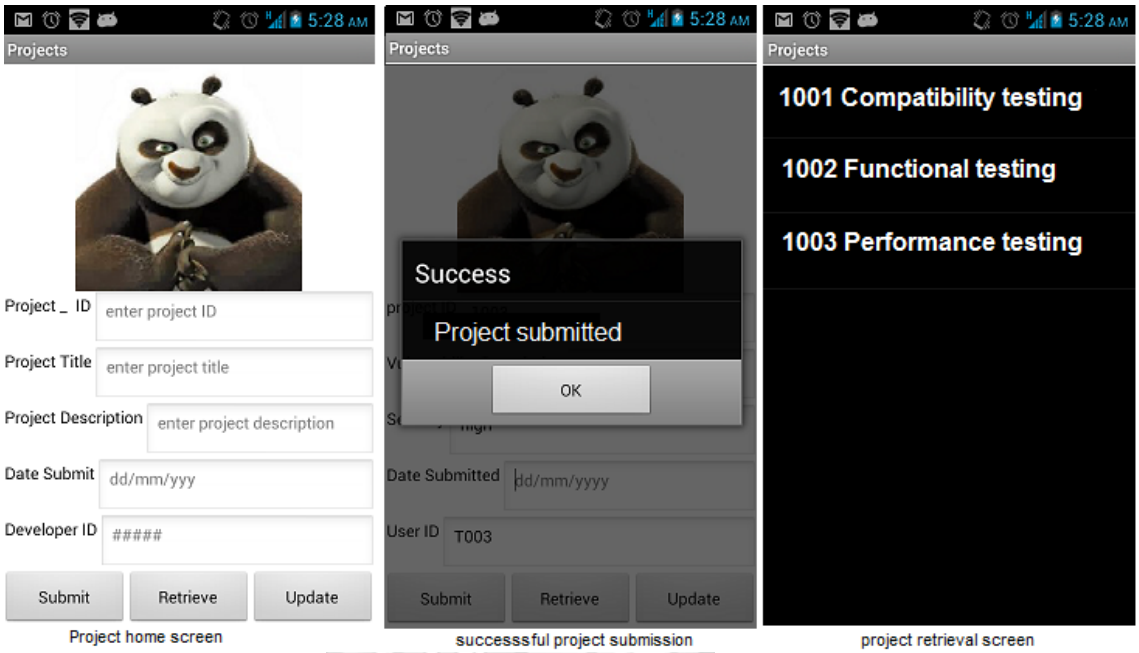

Figure 5.1 project retrieval and display screen

5.2.4.2 Messaging

This module handles submission and retrieval of messages from users. When messages are received from the system, they are displayed in the mobile application screen. A refresh button allows for obtaining of the latest messages. When composing messages a 160 character limit is enforced. See figure 5.2

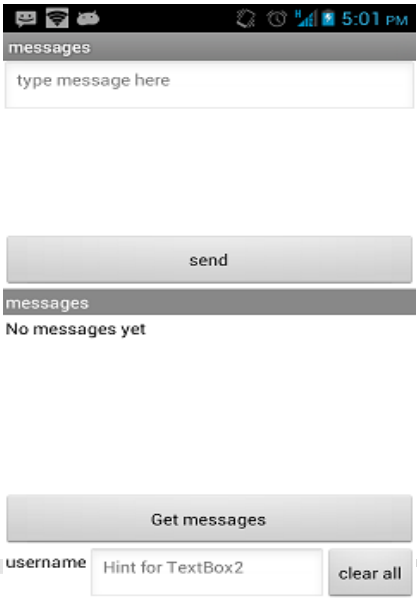

Figure 5.2 messaging screen

5.2.4.3 User Profile Management

The application allows users to create their profiles and to pull their profile pictures from the phone's database. This profile is then submitted to the back end server for storage in the database. See figure 5.3

Figure 5.3 user profile management screen

5.2.4.4 Login and Security

This component shown in figure 5.4 deals with authenticating and authorizing users into the system and works with the backend to validate that usernames and passwords are correct. When a user launches the mobile application they have to key in a valid username and password which is then authenticated in the backend to give a user access. Users who do not have credentials can click on the sign-up button to request for an account.

Figure 5.4 login screen and home screen

5.2.4.4 Bug Report Submission and Retrieval

Vulnerabilities are retrieved from and posted to the web application via HTTP URL calls. The URL contains the report identity number (ID) together with fields that make up a report. The mobile application then populates the phone screen with the report details. See figure 5.5.

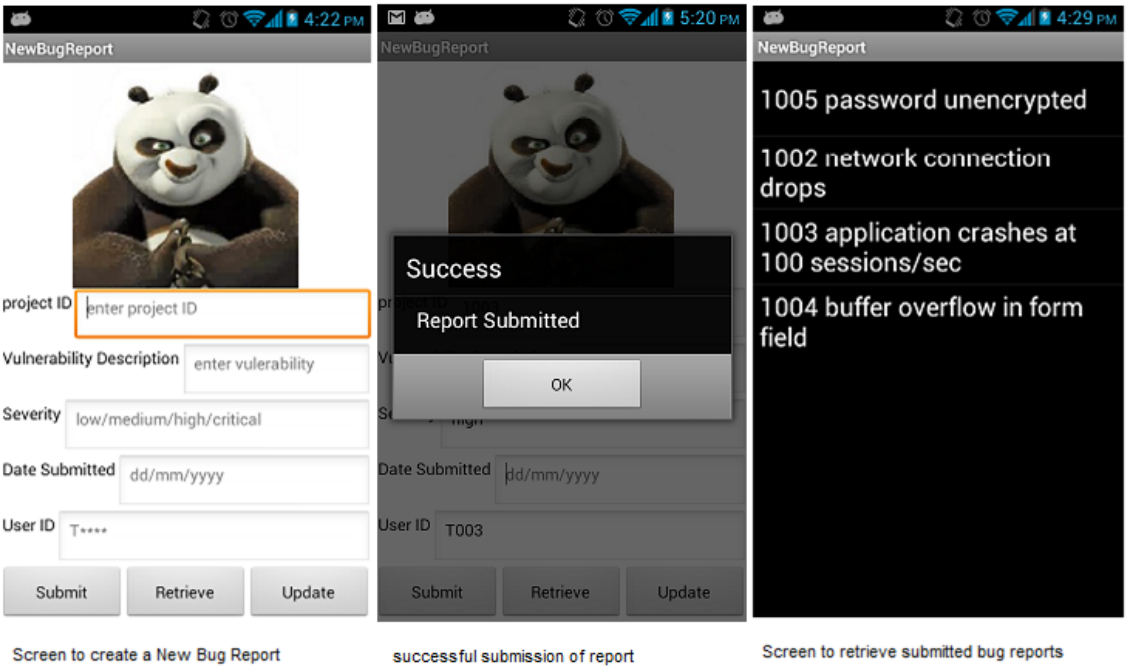

Figure 5.5 bug report screen of the mobile application

5.2.5 Web Application Front and Back End Features

The main components comprising the web front end and back end are shown in figure 5.2 below. The front end features consist of the project view, report creation, messaging and user processing.

Front-end

5.2.5.1 Project View

This module allows for viewing of current projects as well as the creation of new projects. It provides for the extraction of report data in Comma Separated Value (CSV) format. Report details are saved to and retrieved from the database. See figure 5.6

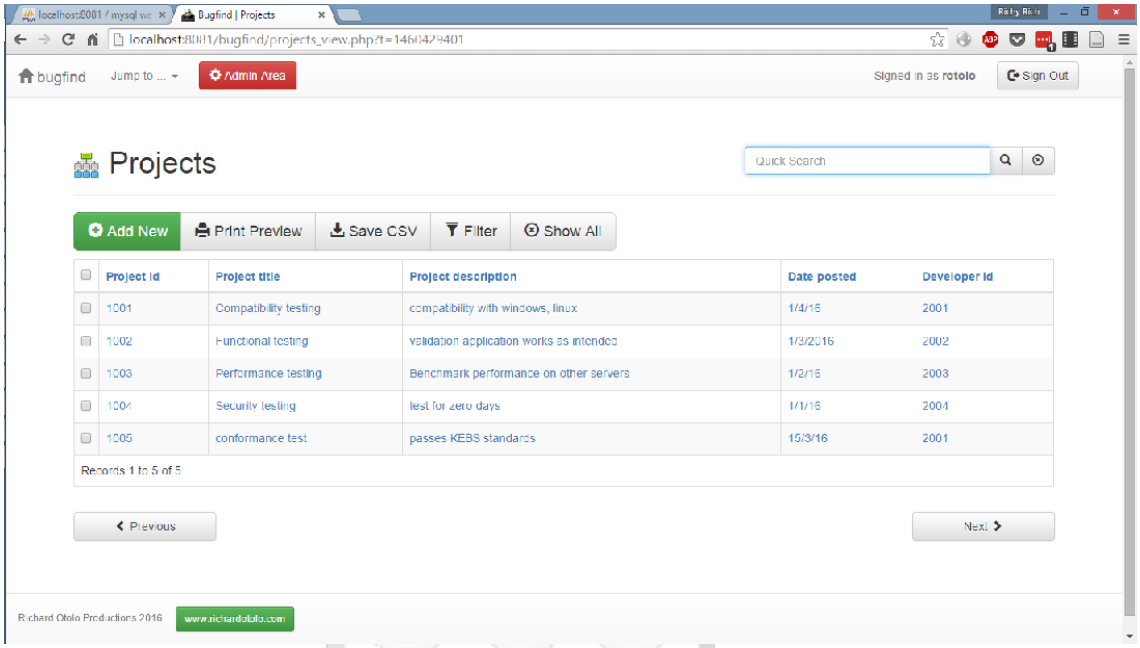

Figure 5.6 web client project screen

5.2.5.2 Report View

This module enables users to view reports that have been submitted by testers. Modifications can be made before onward submission to the database. See figure 5.7.

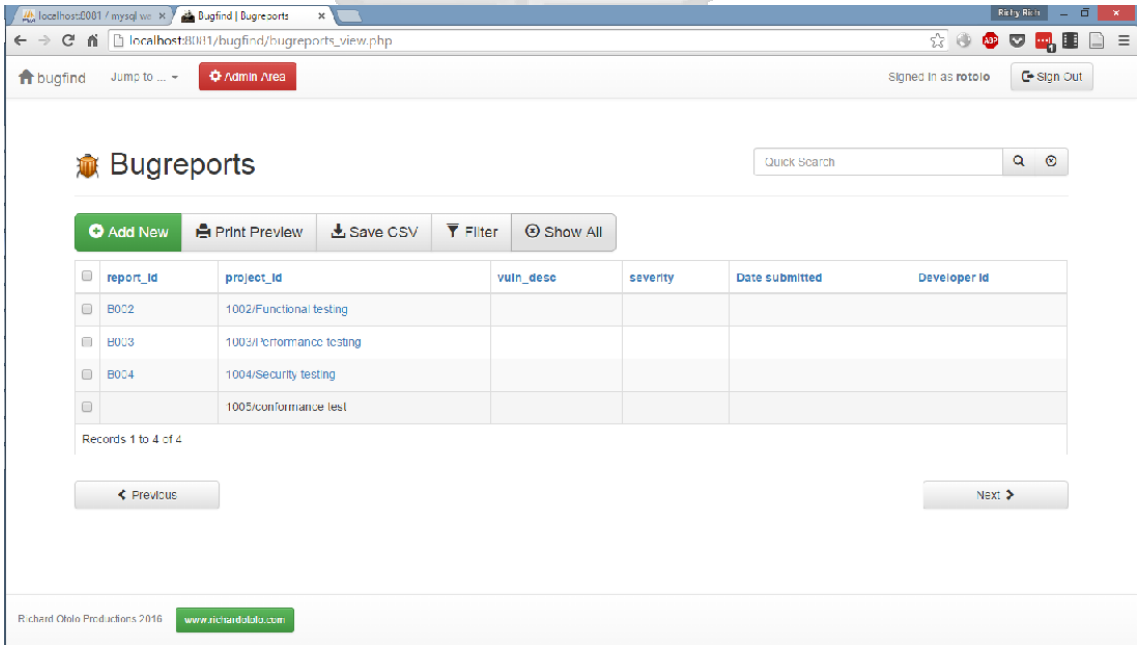

Figure 5.7 web client bug report screen

5.2.5.3 Messaging

This component allows for viewing of messages posted on the platform by various users. Messages are stored in the database as well. See figure 5.8.

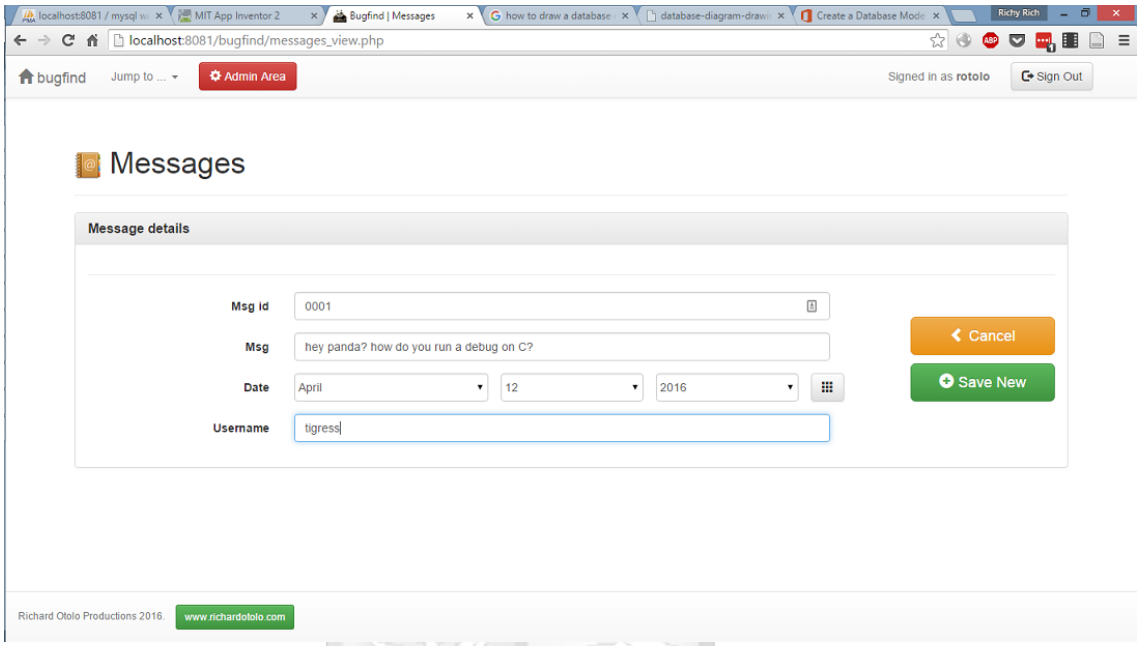

Figure 5.8 web client message report screen

5.2.5.4 User Processing

This module handles the creation of users, updating their details and deleting if necessary. Each user is identified by a unique identification number. See figure 5.9.

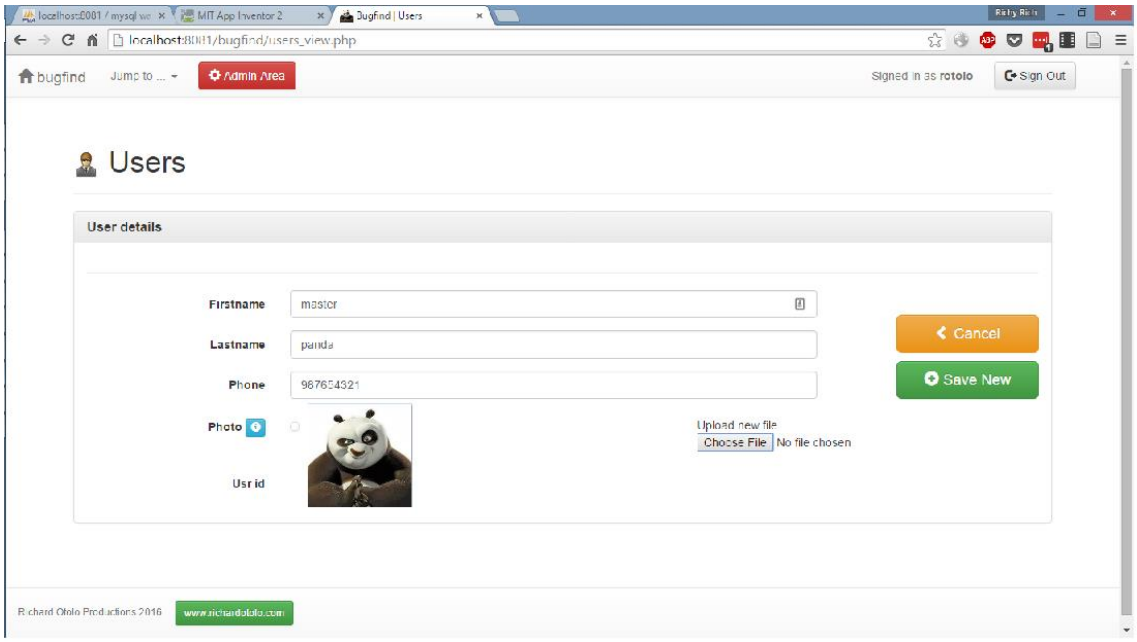

Figure 5.9 web client user profile screen

Back-end

The back end features support the modules found in the mobile application and the web front end.

5.2.5.5Report Processing

This module handles the extraction of records from reports submitted and storing them in the database. It also retrieves

5.2.5.6 Project Processing

This module provides the functions necessary to create new projects before they are submitted to testers as well as stored in the database.

5.2.5.7 Message Processing

This module handles messages generated by end users. It stores these messages in the database and retrieves them when called upon to do so.

5.2.5.8 User Processing

This module supports the creation, deletion, and updating of user details for users in the system.

5.2.5.9 Security

This module prevents unauthorized access to data as well as ensuring the integrity of that data. Access to the mobile, web application and database requires users to have proper credentials. The website makes use of session objects to authenticate and validate users. This removes the need to recheck user validation from the database every time a user moves to another page.

5.2.6 Database

The database provides a repository of data used within the system. It ensures data is served to the various modules connecting to it in a uniform way.

5.3 System Testing

This section describes the various tests carried out on the system developed. The system was evaluated against both functional and non-functional requirements as determined during the evolutionary prototyping phase in Chapter 3 of this dissertation.

5.3.1 Functional Testing

Two kinds of functional tests were carried out. Operational tests were used to verify that the application worked as expected. Additionally, users were asked to rate the suitability of the system as an information sharing tool.

The operational tests on system functionality and the results of those tests are presented in tables 5.1 and 5.2. They provide descriptions of the modules tested, description of the tests conducted, the expected behavior and the observed behavior of the module. The variance column indicates whether there was any deviation between the expected behavior and actual outcome. The think-aloud protocol was recommended where users spoke out what they wanted to do before they did it. This was useful in capturing variance.

Table 5.3 gives the percentage distributions of ratings from respondents assessing various system features. A questionnaire, provided in appendix 1, was employed. It used a Likert scale to allow users to rate the system features. Both types of tests were carried out on the mobile application as well as the web application front end.

Mobile Application Functional Testing				
Test Case	Read a Message			
Description	Test the action of reading a message using the mobile application.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User presses the message icon on the main menu	Messages screen pops out	Messages screen popped out	None	Pass

User presses refresh button	Latest messages are displayed	Latest messages were displayed	None	Pass
Test Case	Read a Message			
Description	Test the action of reading a message using the mobile application.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User presses the compose message button	Compose message screen appears	Compose message screen appeared	None	Pass
User presses the update button	Message sent notification appears.	Message sent notification appeared.	None	Pass
Test Case	Review projects available for selection			
Description	Test the action of viewing available projects using the mobile application.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User presses on the project icon on the home screen	Projects screen is displayed	Projects screen was displayed	None	Pass
Test Case	Make vulnerability report			
Description	Test the action of making a report using the mobile application.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**

User presses submit new bug report icon on the home screen	New bug report screen is presented	New bug report screen was presented	None	Pass
User fills in the fields requiring input and presses enter	Confirmation of report submission and report id are shown	Confirmation of report submission and report id was shown	None	Pass
Test Case	Check Bug report			
Description	Test the action of checking a bug report.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User presses view submitted reports button	Submits bug reports screen is displayed	Submitted bug reports screen is displayed	None	Pass
User presses the submit button without inputting a report ID first	User is prompted to input a report ID	User was prompted to input a report ID	None	Pass
User fills in the fields requiring input and	Report is retrieved	Report was retrieved	None	Pass

presses submit				

Table 5.1 Functional testing of the mobile application.

Web Application Front End Testing				
Test Case	Login/Logout			
Description	Test the action of logging in and out of the web portal.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User logins in with their credentials into the web application.	The user accesses the home page.	The user accessed the home page.	None	Pass
User clicks on the logout button	The user is successfully logged out of their session.	The user successfully logged out of their session.	None	Pass
Test Case	View report			
Description	Test the action of viewing reports on the web portal.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
User clicks on the dashboard icon on the home screen	The dashboard web page is displayed.	The dashboard web page was displayed.	None	Pass
User clicks on the view button	The chosen reports are displayed.	The chosen reports were displayed.	None	Pass
Test Case	Create project			

Description	Test the action of creating new projects			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
The user accesses the projects module and tries to create a new project.	The user can access the projects page and create a new project	The user accessed the project module and was able to create a new project.	None	Pass
Test Case	Manage users			
Description	Test whether the admin can manage all the user entities created in the system.			
TEST	**EXPECTED BEHAVIOR**	**OBSERVED BEHAVIOR**	**VARIANCE**	**COMMENT (pass/fail)**
The admin logins in with their credentials into the web application and access the user details module	The admin is able to access all the users created in the system and is able to create, read, delete and update user details.	The admin was able to carry out the desired tasks successfully	None	Pass

Table 5.2 Functional testing of web application front end.

Table 5.3 below shows the responds rating of various system features.

SYSTEM FEATURE %	**EXTREMELY UNIMPORTANT %**	**SOMEWHAT UNIMPORTANT %**	**MODERATELY UNIMPORTANT %**	**IMPORTANT %**	**EXTREMELY IMPORTANT %**

Ability to have real-time communication between users	0	0	17	33	50
Ability to receive information on reports submitted	0	0	0	50	50
Ability to receive real-time alerts on new projects	0	0	17	66	17
Ability to upload new bug reports from a mobile device	0	17	17	0	66
Ability to select projects from a mobile device	0	0	50	33	17

Table 5.3 Percentages of respondents rating system features

Responses to the open-ended questions are outlined in table 5.4 below.

#	**Question**	**Overall Response**
1	Have you ever used a mobile-based bug reporting tool	100% responded no

2	Why have you never used a mobile-based bug reporting tool?	No knowledge of existing mobile-based bug reporting tools.
3	Have you ever used any other electronic bug reporting system?	66 % responded "Yes"
4	What challenges have you faced with the bug reporting tools you have used?	- Web-based reporting tools do not provide feedback. - Web-based systems are tedious to use - Desktop systems are resource intensive.
5	Have you ever used a crowd-based platform for reporting on software vulnerabilities?	33% responded "Yes"
6	If you answered yes to question 5 above, what challenges did you face when using the bug reporting system on the crowd based platform.	- There is a lack of feedback on the report submitted.

Table 5.4 Responses to the open-ended questions.

5.3.2 Usability Testing

The non-functional testing process focused on the system's usability. Part three of the questionnaire given to respondents (see Appendix 1) sought to collect user feedback on the usability of the application. This goal of testing was to assess how well the system met the six major usability principles namely: layout, content awareness, aesthetics, user experience, consistency, minimal user effort. A Likert scale was used to rate the system.

Feature under test	Strongly Disagree %	Disagree %	Neutral %	Agree %	Strongly Agree %
The information provided on the application is sufficient to allow you to navigate the application without help.	0	66	17	0	17
The application is easy to use and understand how it works	0	33	17	33	17
The design of the application is such that you can predict what each button does.	0	50	0	50	0
The application does not require more than 3 clicks to achieve desired outcomes.	0	17	0	66	17

Table 5.5 Usability test ratings.

The usability tests were conducted and results were collected using a Likert scale. These were assigned weights according to how important each features contribution was. This is shown in table 6.1. Based on the total weights, the features were ranked in order of their importance as seen in Table 6.2.

Rating	Weight
Extremely Unimportant	0
Somewhat Unimportant	1
Moderately Important	2

Important	3
Extremely Important	4

Table 5.6 Weights assigned to rating answers

SYSTEM FEATURE %	EXTREMELY UNIMPORTANT %	SOMEWHAT UNIMPORTANT %	MODERATELY UNIMPORTANT %	IMPORTANT %	EXTREMELY IMPORTANT %	TOTAL WEIGHT
Ability to select projects from a mobile device	0	0	50	33	17	1664
Ability to receive real-time alerts on new projects	0	0	17	66	17	1268
Ability to have real-time communication between users	0	0	17	33	50	1004
Ability to receive information on reports submitted	0	0	0	50	50	800
Ability to upload new bug reports from a mobile device	0	17	17	0	66	468

Table 5.7 Ranking of responses according to total weight

CHAPTER 6 DISCUSSIONS AND KEY FINDINGS

This chapter presents an analysis of the results of work done in the previous chapter and demonstrates how the research objectives that were outlined in chapter 1 were met.

6.1 Research Findings

6.1.1 Acceptability, Operation, and Effectiveness

This section discusses the results of functional testing carried out on the mobile and web application. These tests were based on comparing the variance in expected behavior when a typical tester used the system against the observed behavior.

From the results of the functional tests, it was evident that both the mobile and web application worked as intended. There was zero variance between anticipated behavior and observed behavior. The respondents indicated that the system could be effectively used for the exchange of real-time information between testers and developers in a crowdsourced software testing platform.

6.1.2 Usability Test Analysis

This section discusses the perceptions of respondents that were measured using a Likert scale. The results were obtained from a total of six respondents.

Figure 6.1 below shows the perception of respondents on the importance of being able to select projects from their mobile devices.

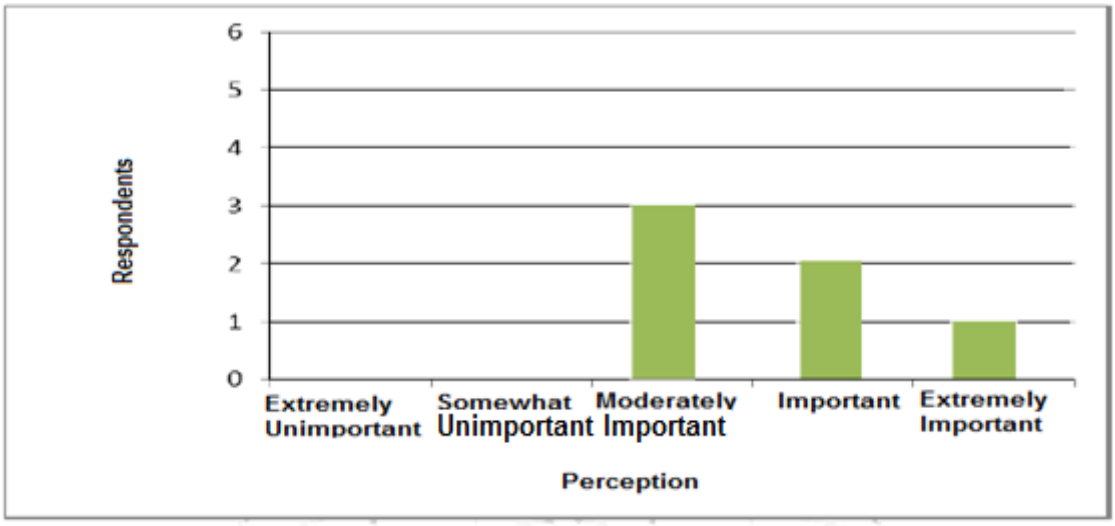

Figure 6.1 Ability to select projects from a mobile device

Most of the respondents felt it was not a very important feature to be able to select projects from the mobile device as they could do the same on the testing platform.

Figure 6.2 below shows the perception of respondents on the ability to receive real-time alerts on new projects.

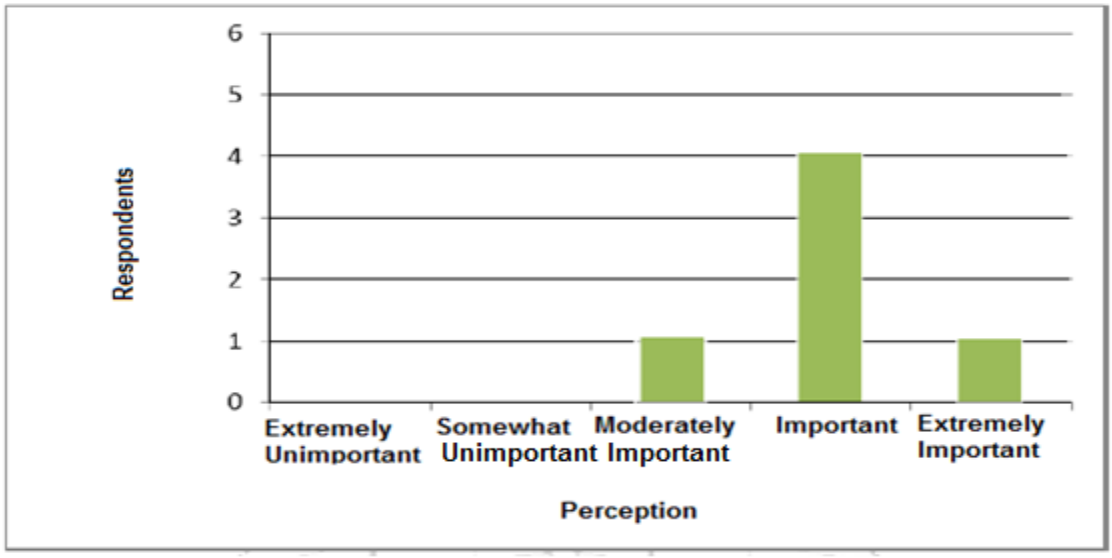

Figure 6.2 Ability to receive real-time alerts on new projects

All of the respondents felt it was important to be able to receive information on reports on their mobile devices. This was because they were able to receive updates on any new projects at any time and place they might be.

Figure 6.3 shows the feedback from respondents on their perception of the ability to have real-time communication between users using the mobile application.

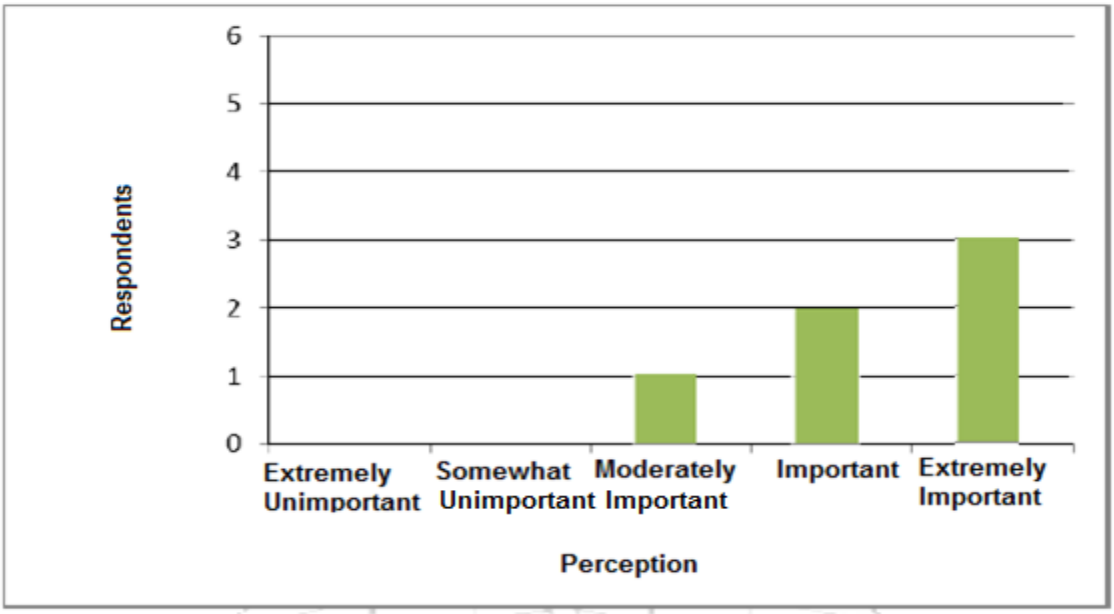

Figure 6.3 Ability to have real-time communication between users

The majority of the respondents felt it was extremely important to have the ability to communicate with their fellow testers in real-time. This was because it allowed them to be able to assist each other as well as exchange ideas which lead to better quality work.

Figure 6.4 depicts the perception of respondents on the importance of being able to receive information on reports that have been submitted.

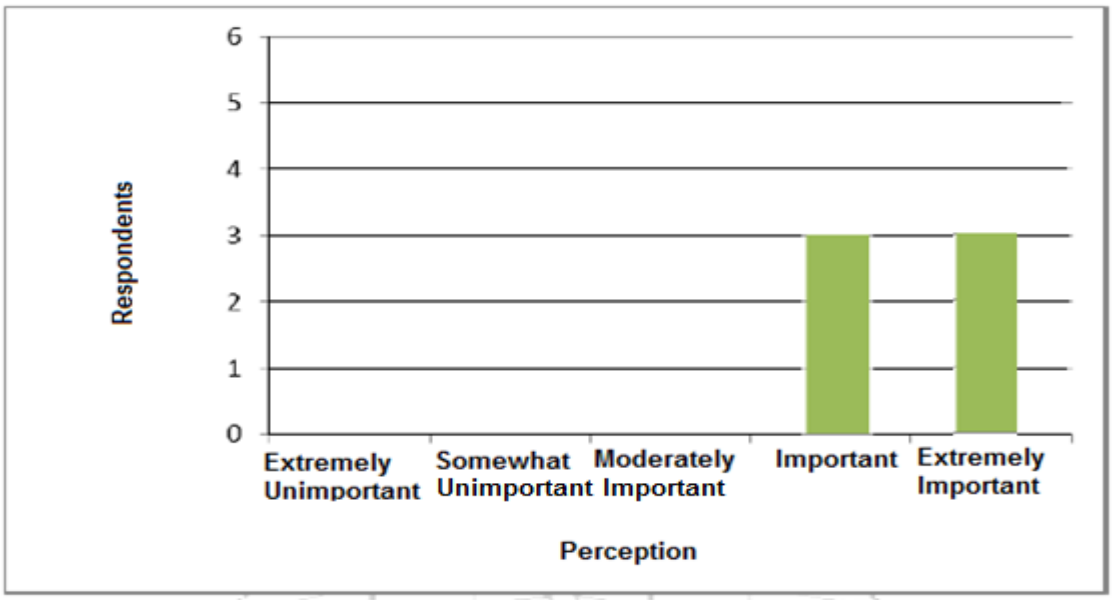

Figure 6.4 Ability to receive information on reports submitted

The respondents were unanimous that receiving instant feedback on the reports they had submitted was crucial as they would be able to quickly calculate know how much money they would be paid for their work.

Figure 6.5 shows the results collected for respondent's perception of the importance of being able to upload new vulnerability reports from their mobile devices.

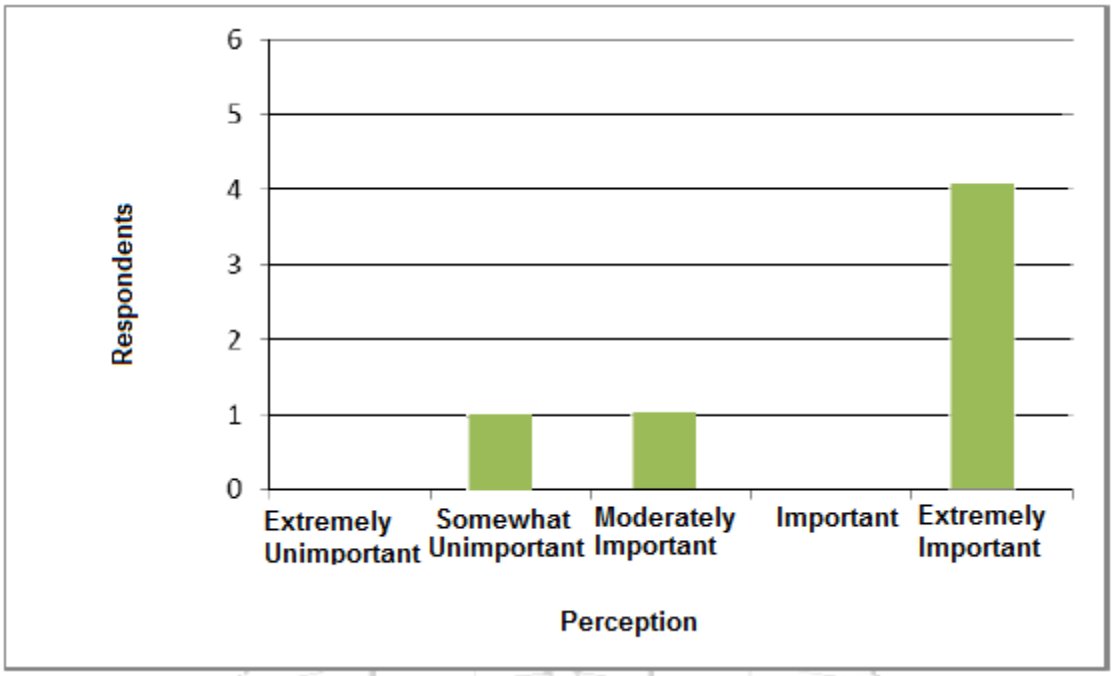

Figure 6.5 Ability to upload new bug reports from a mobile device

Majority felt that this was extremely important because it would allow them to respond quickly and thus increase their chances of being the first to report on a bug they had found.

6.2 Relevance of the Research Work to the Research Objectives

6.2.1 Relevance to Research Objectives

The research objectives and questions provided the framework that guided this research as well as the development of the proposed system.

The first objective sought to determine the sources of security vulnerabilities in software. This objective was achieved by performing a literature review of software development methodologies. The main conclusions were that vulnerabilities came about when developers failed to factor in security during application design, this could be as a result of lack of the prerequisite skill, malicious intent or carelessness. Thus, if a proper way of testing applications before they are released is put in place the number of vulnerabilities found in the final product will decrease.

The second objective sought to identify the weaknesses that existing crowd-sourced application testing systems have in sharing of test-related information. This objective was achieved by reviewing the literature on such systems as well by surveying developers who have interacted with such systems. It emerged that the main weaknesses came from a lack of a real-time and on-demand method of communication between stakeholders. Hence, the reason this study proposed the use of a mobile-based system to improve on this communication.

The third objective was to develop a mobile-based prototype that will improve on how test-related information is shared on crowd-sourced application testing systems. Evolutionary prototyping was used to achieve this objective as discussed in Chapter 3. The use of prototyping was beneficial because it helped to quickly communicate the visual aspects of the proposed system. Additionally, users were able to interact with the system and provide feedback that changed the researcher's perspective on the importance of certain modules in reporting. The system architecture and design were discussed in chapter 4 and screenshots of the developed system are provided in appendix B. Two major tests were carried out, namely: functional and non-functional testing and from the results presented in chapter 5 it was seen that the prototype functioned as expected.

The fourth objective aimed to evaluate the effectiveness of the developed prototype as far as sharing of test-related information was concerned. From the responses obtained using the Likert scale presented in section 5.3.4, it can be seen that the features testers and developers found most useful in an information sharing tool were: the ability to receive real-time alerts on new projects, to receive projects and being able to communicate in real-time with other users on the platform.

6.2.2 Relevance to Research Questions

Table 6.3 below summarizes how the research questions were addressed. Q1, Q2, Q3, and Q4 have been used to represent the research questions and they are mapped as below.

Q1. What are the sources of security vulnerabilities in software applications?

Q2. What gaps do existing crowd-sourced application testing systems have in terms of sharing test-related information?

Q3. How can a prototype that addresses information sharing gaps in existing crowd-sourced software testing platforms be designed?

Q4. How effective is the developed prototype in improving the sharing of test-related information?

	Literature review (see Chapter 2)	Questionnaire/Prototype test (see Appendix 1 and Chapter 5)	Design and Implementation (see Chapter 3, 4 and 5)
Q1	Review literature to investigate sources of software vulnerabilities	Ask developers for their experience with handling software vulnerabilities.	
Q2	Review literature and websites on existing crowd-testing systems.	Ask developers who have used existing crowd-testing systems for their experience with them.	
Q3	Review literature and websites on features of existing crowd-testing systems and identify the challenges they face.		Analyze, design and implement a prototype that seeks to addresses the challenges identified in existing crowd-testing systems.
Q4		Run a survey to obtain feedback from respondents on the suitability of the proposed prototype for reporting. Carry out a pilot using the	

		software to get feedback to improve it.	

Table 6.1 Mapping of Research Questions to Research Methods.

6.3 Conclusion

This chapter has demonstrated how the research objectives and research questions were successfully addressed. It has also shown that respondents found the developed system to be effective sharing of test-related information.

CHAPTER 7 CONCLUSIONS AND RECOMMENDATIONS

7.1 Introduction

This chapter presents a summary of the research work conducted and proposes some recommendations and future work to be undertaken.

7.2 Conclusions

At the beginning of this research, a background study was conducted which confirmed that the mode of sharing of test-related information in crowd testing systems needed improvement. Based on the weaknesses in the system, research questions and objectives were set out with the aim of developing a mobile solution that would enable testers and developers better able to share information about projects, reports, in real-time. To support these objectives, a literature review was conducted to study a number of existing systems.

In chapter 3 a prototype was developed and exposed to potential end users with the aim of getting feedback on improvements that needed to be made. In chapter 4 the system was designed using common tools found in software engineering such as use cases, data flow diagrams etc. The system was implemented and screenshots provided in chapter 5 of this dissertation. Functional and usability tests were conducted and results presented in the same chapter. Chapter 6 provided a discussion on how the research objectives and questions were addressed.

7.3 Recommendations

In order to implement this system, in reality, companies that offer crowd-sourced testing services would have to add additional modules to their platforms to facilitate the use of communication via mobile phones. This would also mean re-engineering current test information sharing processes to accommodate mobile technology. Additionally, it would be a requirement that testers and developers who want to use the system to have a smartphone.

7.4 Suggestions for Future Work

The mobile application developed in this dissertation allowed users to provide text-based information. It was suggested that a way be found to provide additional ways of communication such as video and audio. With the ongoing improvement in Internet bandwidth infrastructure and reduction in the cost of Internet access, this is a feasible project in the immediate future.

Furthermore, the mobile application was developed to run on an android operating system. Given that there are many other types of smartphone operating systems in use such as Apple's iOS, Windows mobile etc., it would be beneficial to develop an application that runs on these platforms.

REFERENCES

Bugcrowd. (2016). Retrieved March 14, 2016, from https://bugcrowd.com/

Bugzilla. (n.d.). Retrieved March 20, 2016, from https://www.bugzilla.org/

Carter, R. A., Anton, A. I., & Williams, L. A. (2001). Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model., 94–101. http://doi.org/10.1109/ISRE.2001.948548

Common Attack Pattern Enumeration and Classification. (2016). Retrieved March 20, 2016, from http://capec.mitre.org/

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, *14*(10), 1453–1461. http://doi.org/10.1109/32.6190

Gupta, A. K., Chandrashekhar, U., Sabnis, S. V., & Bastry, F. A. (2007). Building secure products and solutions. *Bell Labs Technical Journal*, *12*(3), 21–38. http://doi.org/10.1002/bltj.20247

HackerOne. (2016). Retrieved March 14, 2016, from https://hackerone.com/

Hein, D., & Saiedian, H. (2009). Secure Software Engineering: Learning from the Past to Address Future Challenges. *Information Security Journal: A Global Perspective*, *18*(1), 8–25. http://doi.org/10.1080/19393550802623206

Jacobson, I., Jonsson, C., & Overgaard, G. (1992). *Object-oriented software engineering - A use case driven approach*. Addison-Wesley.

Khan, M. (2011). Software Vulnerability: Basic Concepts, Tools, and Research Survey.

Krsul, I. V. (1998). *Software Vulnerability Analysis*. Purdue University, West Lafayette, IN, USA.

Kumar, R. (2011). *Research Methodology: a step by step guide for beginners* (3rd ed.). sage publications.

Labaree, R. V. (2016, April). Research Guides: Organizing Your Social Sciences Research Paper: Types of Research Designs. Retrieved April 12, 2016, from http://libguides.usc.edu/writingguide/researchdesigns

Lipner, S. (2004). The Trustworthy Computing Security Development Lifecycle. *20th Annual Computer Security Applications Conference '04*, *1063-9527*.

Mao, K., Capra, L., Harman, M., & Jia, Y. (2015). *A Survey of the Use of Crowdsouring in Software Engineering. Technical Report* (No. RN/15/01). London: Department of Computer Science, University College London.

McGraw, G. (2004, March 31). What is Software Security? Retrieved from https://www.cigital.com/blog/software-security/

Mellacheru, P., & Swaminathan, A. (2014). Harnessing the Power of the Crowd for the Mob: A guide to crowdsourced mobility testing for BYOD enabled enterprises.

Microsoft Security Response Center. (2016). Retrieved March 20, 2016, from https://technet.microsoft.com/en-us/security/dn440717

MITRE Corporation. (2015a). Common Vulnerabilities and Exposures. Retrieved from https://cve.mitre.org

MITRE Corporation. (2015b). CWE - Common Weakness Enumeration. Retrieved March 20, 2016, from http://cwe.mitre.org/

Mozilla Foundation Security Advisories. (2016). Retrieved March 20, 2016, from https://www.mozilla.org/en-US/security/advisories/

NASA. (n.d). A Predictive Approach to Eliminating Errors in Software Code. Retrieved March 20, 2016, from https://spinoff.nasa.gov/Spinoff2006/ct_1.html

National Vulnerability Database. (2015). Retrieved March 20, 2016, from https://nvd.nist.gov/

Noopur, D. (2005). *Secure Software Development Life Cycle Processes: A Technology Scouting Report* (Technical Note). Carnegie Mellon University.

Open Source Vulnerability Database. (2013, March 13). Retrieved from https://blog.osvdb.org/about/

Ponemon Institute. (2012). *2012 Application Security Gap Study: A Survey of IT Security & Developers* (p. 24). Ponemon Institue LLC.

Ruparelia, N. B. (2010). Software Development Lifecycle Models. *SIGSOFT Softw. Eng. Notes*, *35*(3), 8–13. http://doi.org/10.1145/1764810.1764814

Secunia. (n.d.). Retrieved March 20, 2016, from http://secunia.com/

SecurityFocus. (n.d.). Retrieved March 20, 2016, from http://www.securityfocus.com/about

Shen, X. (2015). Mobile crowdsourcing [Editor's note]. *IEEE Network*, *29*(3), 2–3. http://doi.org/10.1109/MNET.2015.7113217

Sindre, G., & Opdahl, A. L. (2008). Misuse cases for identifying system dependability threats. *Journal of Information Privacy and Security*, *4*(2), 3–22.

Steinhauser, M. (2013, November). Two approaches to Crowdsourced Software Testing.

Synack. (2016). Retrieved March 14, 2016, from https://www.synack.com/

Thakral, P., & Sharma, S. (2014). Comparison of Software Life Cycle Models. *International Journal of Software and Web Sciences*, *10*(1), 73–76.

Tsipenyuk, K., Chess, B., & McGraw, G. (2005). Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. *IEEE Security and Privacy*, *3*(6), 81–84. http://doi.org/83900927-3F85-4A31-B0A7-5EDD585CF7F3

US-CERT | United States Computer Emergency Readiness Team. (n.d.). Retrieved March 20, 2016, from https://www.us-cert.gov/

Whitfield, K. (2013, April). What Apps are People Using? Portio Research.

Zhang, X., Chen, Z., Fang, C., & Liu, Z. (2016). Guiding the Crowds for Android Testing.

Zogaj, S., & Bretschneider, U. (2014). Analyzing governance mechanisms for crowdsourcing information systems: A multiple case analysis. Retrieved April 9, 2016, from http://pubs.wi-kassel.de/wp-content/uploads/2015/04/JML_531.pdf

Zogaj, S., Bretschneider, U., & Leimeister, J. M. (2014). Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. *Journal of Business Economics*, *84*(3), 375–405. http://doi.org/10.1007/s11573-014-0721-9

APPENDICES

Appendix A: Questionnaire

This questionnaire is part of a study: An Information Sharing Tool for Crowd-sourced Software Testers. It aims to investigate end user perceptions towards the developed mobile application. The information used in this questionnaire will be treated in confidentiality and will be used for academic purposes only.

Part 1: Personal Profile

1. **Respondents name**:
2. **Respondents occupation**:

Part 1: General

1. Have you ever used any mobile-based vulnerability reporting tool before?	
2. If the answer to 1 was NO, why not?	
3. Have you ever used any other vulnerability reporting tool	
4. What challenges have you faced with the vulnerability reporting tools you have used?	
5. Have you ever used a crowd-based platform for reporting bugs you found in the software you were testing?	
6. What challenges did you find with the information sharing system used by the crowd-based platform?	

Part 2: Features

The mobile-based system contains the following features. Please rate each feature in terms of how important it is by ticking where appropriate. Whereby EU-Extremely Unimportant, SU-Somewhat Important, MI-Moderately Important, EI Extremely Important

	EU	**SU**	**MI**	**I**	**EI**

1. The system allows for real-time communication with other users					
2. The system allows users to receive real-time information about reports that have been submitted.					
3. The system provides real-time alerts on new projects.					
4. The system allows for uploading of bug reports from a mobile device.					
5. The system allows you to pick projects from the mobile device					

Part 3: User Experience

Please tick where appropriate whereby SD – Strongly Disagree, D – Disagree, N – Neutral, A – Agree, SA – Strongly Agree

	SD	**D**	**N**	**A**	**S**
1. The information provided on the application is sufficient to allow you to navigate the application without help.					
2. The application is easy to use and understand how it works					
3. The design of the application is such that you can predict what each button does.					
4. The application does not require more than 3 clicks to achieve desired outcomes.					

Table A1 Questionnaire

Apeendix B: Turn-it-in Report

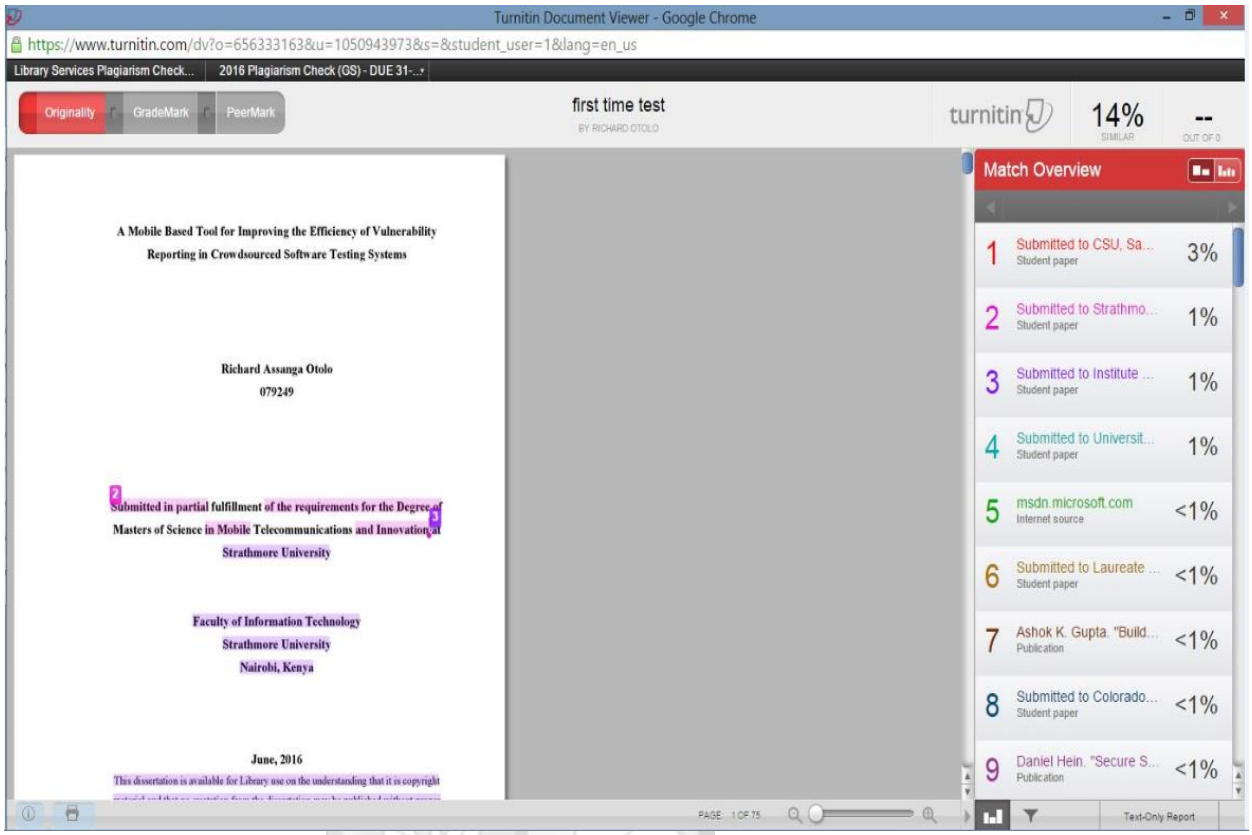

Figure A2 Turn-it-in report