SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Date: October 9, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Thomas King
Samantha Lee
Madison Martin

ENTITLED

Bug Reporting System

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER ENGINEERING

_____
Thesis Advisor

_____
Thesis Advisor

_____
Department Chair

_____
Department Chair

Bug Reporting System

by

Thomas King
Samantha Lee
Madison Martin

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Engineering
School of Engineering
Santa Clara University

Santa Clara, California
October 9, 2018

Bug Reporting System

Thomas King
Samantha Lee
Madison Martin

Department of Computer Engineering
Santa Clara University
October 9, 2018

ABSTRACT

Currently, there aren't adequate bug detecting systems for Santa Clara University's IT department to efficiently track and eliminate bugs within the system. Our project's objective is to provide a solution to aid with the process of finding, reporting, testing, eliminating, and mitigating present and future bugs. Within the system there are four active user types: reporter, tester, developer, and manager. Each user is able to access specific webpages through their account they can log in and log out of. Reporters can report a bug and check on the status of the bugs they report. Testers and developers are able to check the status of all bugs, test bugs and solutions, assign bugs to other testers and developers, and update the status of bugs they're working on. Managers can also check the status of the bugs, assign bugs, and update the statuses of bugs. Reports, history, and updates on the bugs are inputted through the web page forms on our site and can be organized by priority or frequency. Information will be kept in a database using MySQL. The backend of our site will run using Python, specifically the Flask framework. We will be using a client-server architecture with a server connected to a database to allow multiple clients to access the services at the same time.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Bugs happen. When bugs in code occur in regularly used software, the unexpected results can hinder productivity. In a school setting, if the online websites for students do not work as intended, a student could not register for their desired courses, get the study materials their professors put online, submit an assignment on time, or use other functions.

The SCU community should be able to report bugs in real time and have those issues follow a chain of command so they can be fixed by the proper people in a timely manner. When someone using SCU software encounters a bug, the only way to report it is to talk to the IT Department directly, which is too much effort for most users, so the bugs remain unreported. The only other time bugs are fixed is when they are found by the developers or testers. Providing an easy method for user feedback allows more bugs to be found earlier so they can be fixed before affecting too many people.

1.2 Solution

Our solution is to create a website that would enable people with an SCU login to report bugs on SCU applications. The solution will consist of a structured flow of web pages allowing the testers, managers, and developers to share feedback and update the status of a given bug while adding notes on the specifics of the bug as well as the severity and the time to completion. There will also be a searchable history feature that will help with future bug mitigation by congregating all previous bugs with the steps taken to reproduce it and the solution which can facilitate a faster repair the next time a similar bug happens. The web pages will communicate with a server which will save information like bug information and status, email addresses for notifications, and account information for authentication, in a database. When a new session is launched the web page will retrieve relevant information from the server.

Chapter 2

Requirements

2.1 Functional

The functional and non-functional requirements, as well as the design constraints, are listed below in order to define the needs of the system. Critical requirements are those that are necessary, and recommended are additional features that could be added given enough time.

2.1.1 Critical

The system will allow a user to...

- Authenticate using their SCU account
- Report bugs into the system
- Check on status of previous bugs
- Assign bugs to qualified developers (if manager)
- View overall reports (if manager)
- Change the status of a bug (if tester or developer)

2.1.2 Recommended

It is recommended to allow...

- Reporters to get an email confirmation after submitting their bug
- Testers and managers to keep track of duplicate bugs
- Users to search through the history of bugs

2.2 Nonfunctional

2.2.1 Critical

It is required that the web application is...

- Visually appealing and not cluttered with too much information on any single page
- Easy to use for each of the roles
- Easy to expand with new features
- Updated frequently with the status changes of the bugs

2.2.2 Recommended

It is recommended that the system is...

- Useful to everyone with a SCU account
- Modular and easy to expand to other platforms
- Up to date with the most recent standards

2.3 Design Constraints

The system must...

- work on the Linux machines in the Design Center
- use SCU accounts for authentication

Chapter 3

Use Cases

This system has four different roles: reporter, tester, manager, and developer. A reporter is a user that reports a bug in the system. A tester is a person who reproduces reported bugs and tests bug fixes. A manager is the individual who assigns a bug to a developer. A developer is the person who fixes the bug and once the fix is completed, sends it back to the tester to make sure their solution is working.

3.1 Use Case Diagram

Figure 3.1 below demonstrates the overlap of actions each type of user can perform in the system.

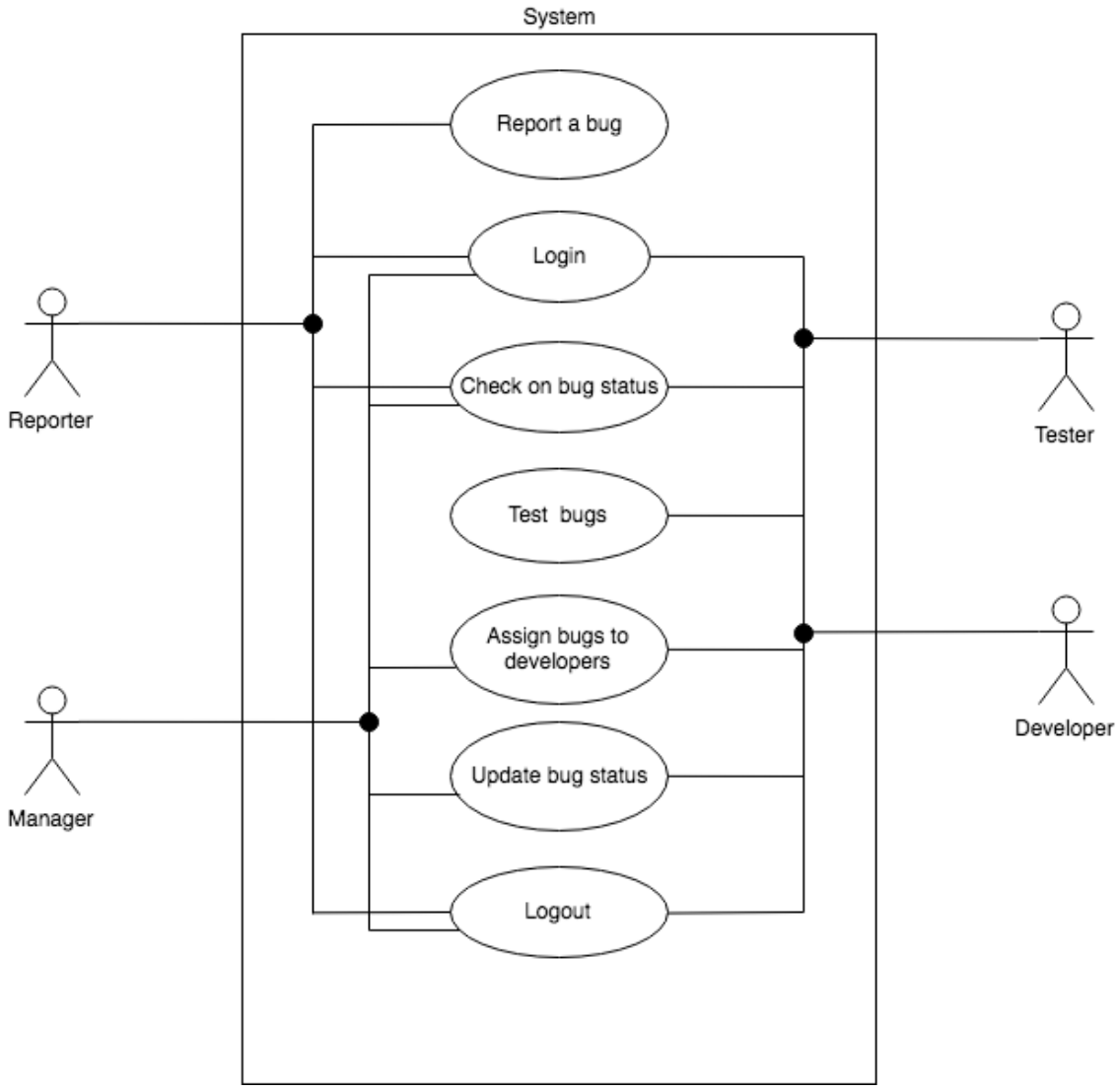

Figure 3.1: Use Cases Diagram

3.2 Cases

- **Log in**

  Goal: Log in using an SCU account to get access to the report page and the bug status page.

  Actors: Reporter, Tester, Manager, Developer

  Preconditions: User must have an SCU account and a computer with a working internet connection.

  Postconditions: User is notified if the log in was successful.

Exceptions: None

- **Report a bug**

  Goal: Fill out the report form and submit it so that the experienced bug will be fixed.

  Actors: Reporter

  Preconditions: Reporter must have logged in successfully.

  Postconditions: Reporter is notified if the bug submission is successful.

  Exceptions: None

- **Check on bug status**

  Goal: Find the desired bug and check on its status to see if it is fixed or not.

  Actors: Reporter, Tester, Manager, Developer

  Preconditions: User must have logged in successfully.

  Postconditions: Table displays the history of all bugs that have been reported and their status.

  Exceptions: None

- **Test bugs**

  Goal: Verify the state of the bug

  Actors: Tester, Developer

  Preconditions: Description of bug must be submitted through the reporter form.

  Postconditions: None

  Exceptions: None

- **Assign bugs to developers**

  Goal: Designate a bug to fix to a qualified developer.

  Actors: Manager

  Preconditions: Manager must have logged in successfully and there are bugs that need to be assigned.

  Postconditions: Developer is notified of a new bug to fix.

  Exceptions: None

- **Update bug status**

  Goal: Change the status of the bug depending on what part of the fixing process it is at.

  Actors: Tester, Manager, Developer

  Preconditions: Manager must have logged in successfully.

Postconditions: Status value of the bug reflects the inputted value.

Exceptions: None

- **Log out**

  Goal: After done viewing the desired pages, the user will log out of their account.

  Actors: Reporter, Tester, Manager, Developer

  Preconditions: User must have logged in successfully.

  Postconditions: User is notified that they have logged out of the system.

  Exceptions: None

Chapter 4

Activity Diagrams

Figures 4.1-4.4 describe the activity flow for each user type: reporter, tester, developer, and manager.

4.1 Reporter

A reporter can report a bug and check the status of their previous bug reports.

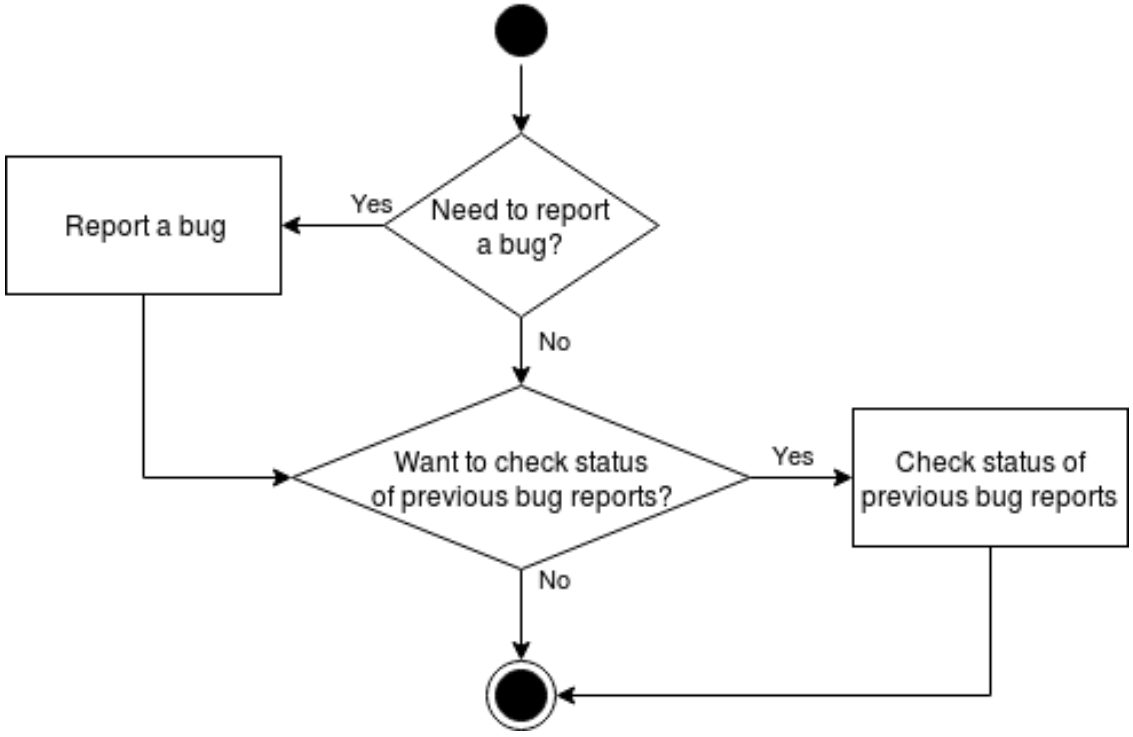

Figure 4.1: Reporter Activity Diagram

4.2 Tester

A tester can mark bugs as duplicates, reproduce bugs, and test bugs to confirm fixes.

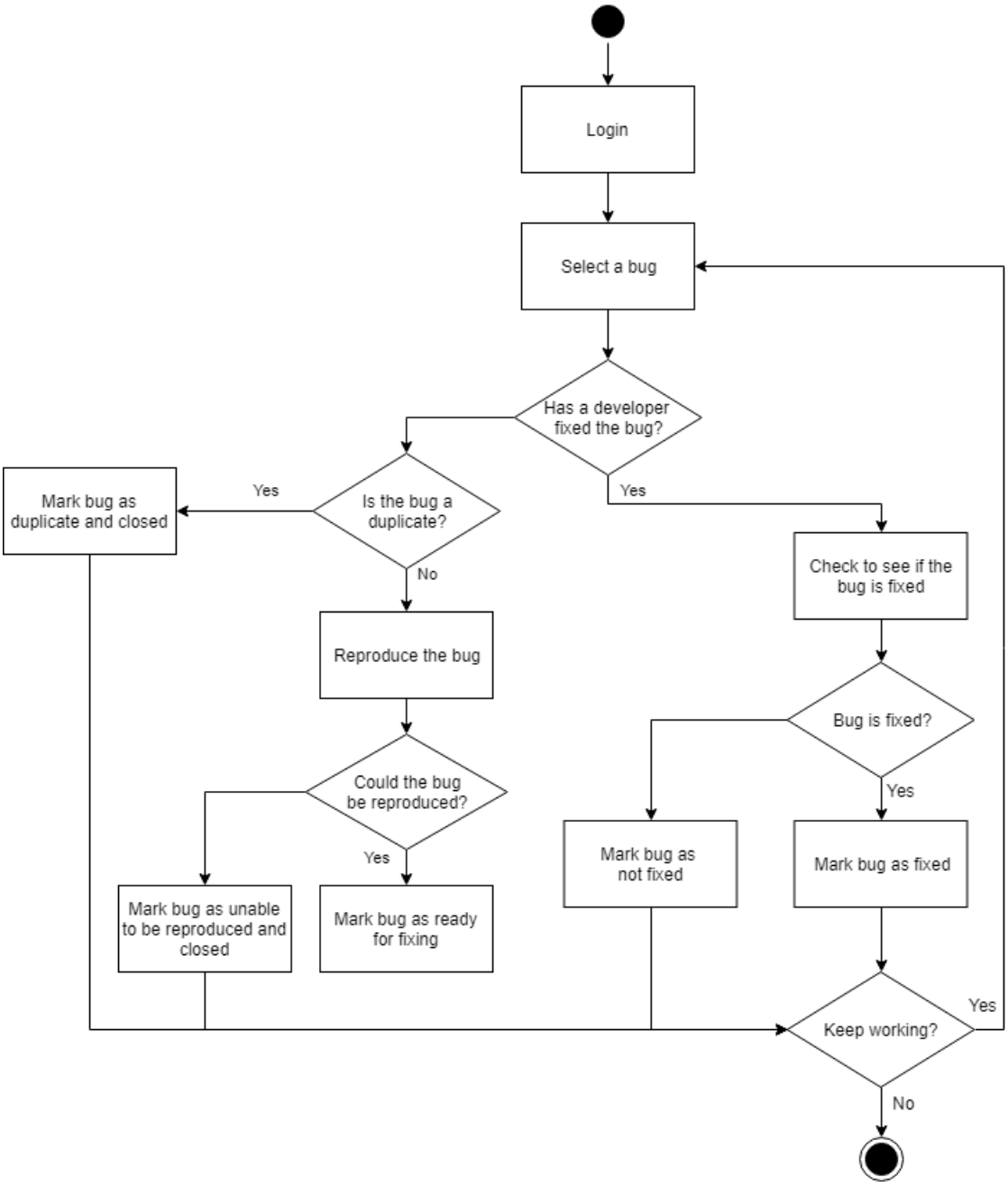

Figure 4.2: Tester Activity Diagram

4.3 Developer

A developer can fix bugs and mark them as ready for testing.

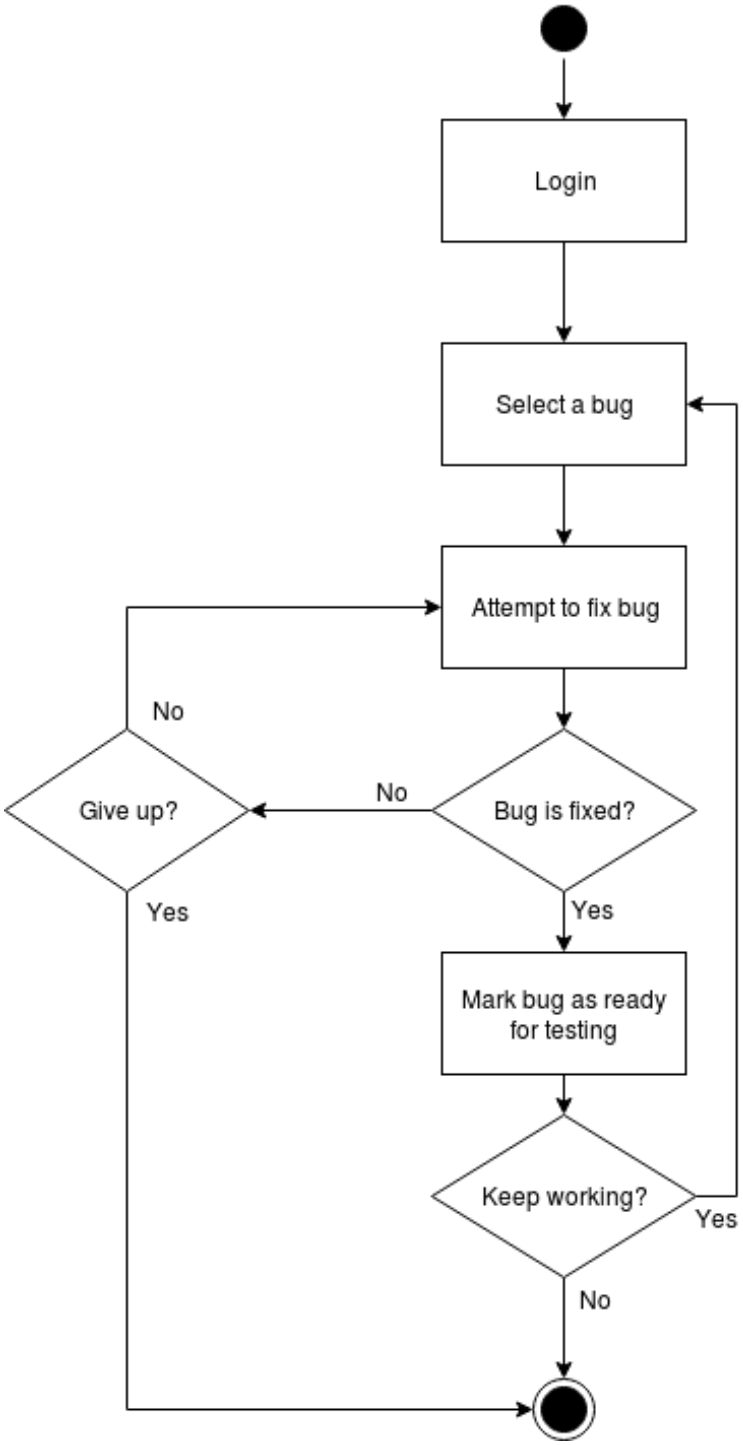

Figure 4.3: Developer Activity Diagram

4.4 Manager

A manager can look at reports about the system and assign reproduced bugs to developers.

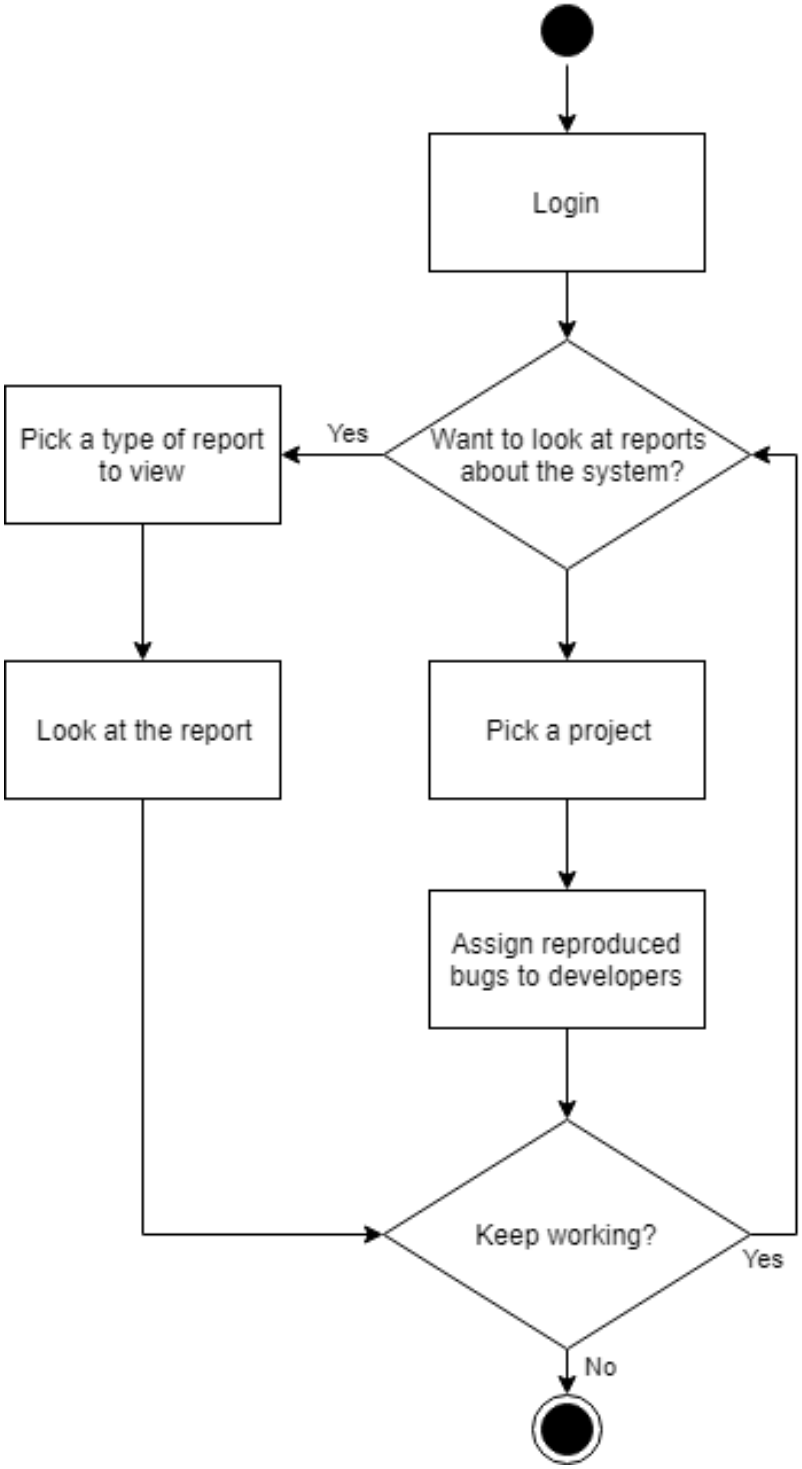

Figure 4.4: Manager Activity Diagram

Chapter 5

Conceptual Model

Figures 5.1-5.6 demonstrate the windows the user will experience given their particular role. Since this website will be accessible only after going through SCU authentication, all of the models are visited after the log in.

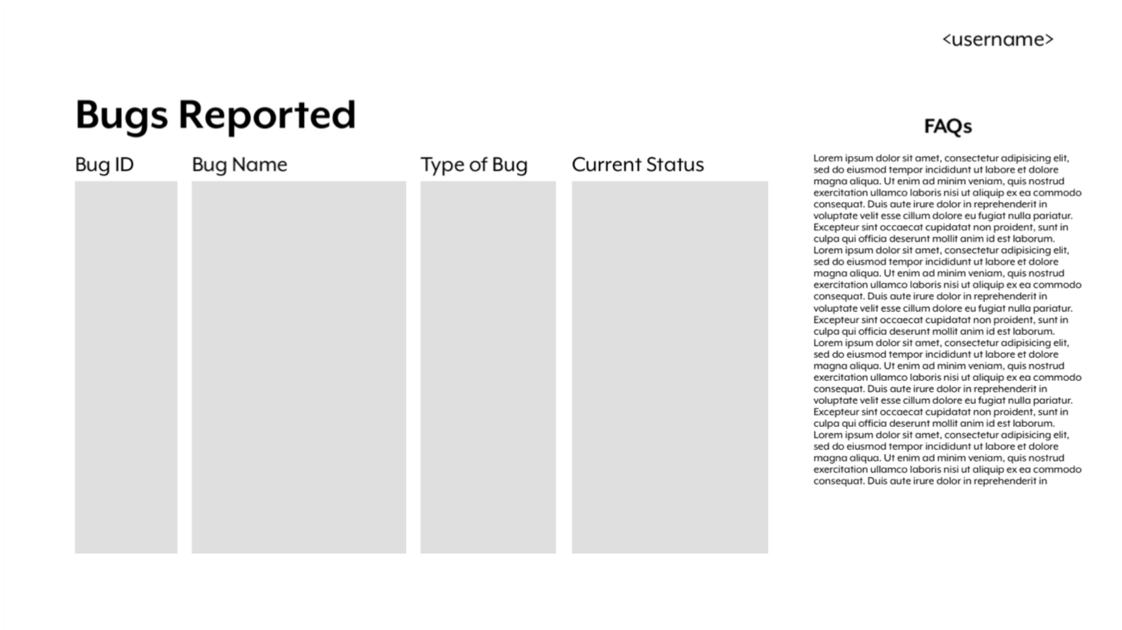

Figure 5.1: Bug History page

Figure 5.2: Pop up window that displays a more detailed description of an individual bug

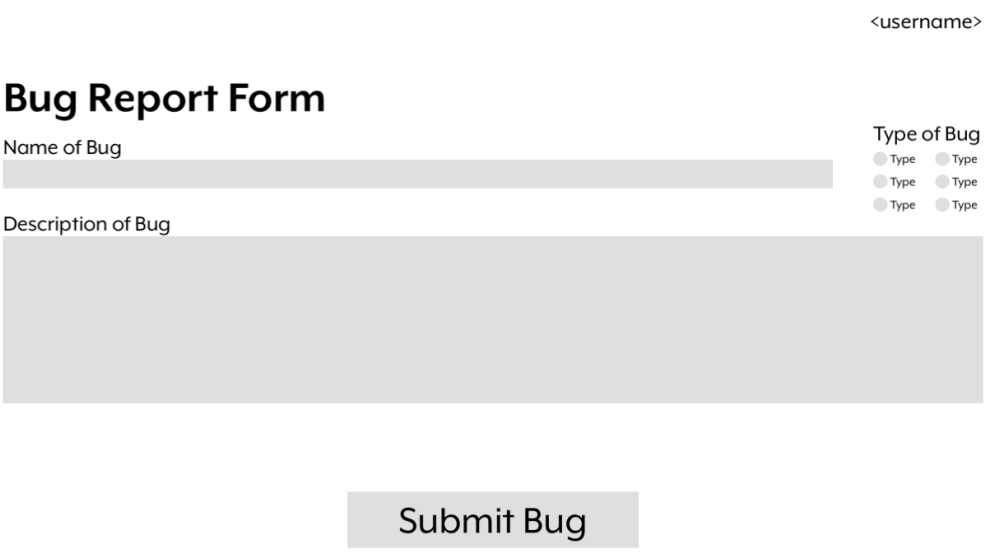

Figure 5.3: Bug Report page that a reporter will be able to access

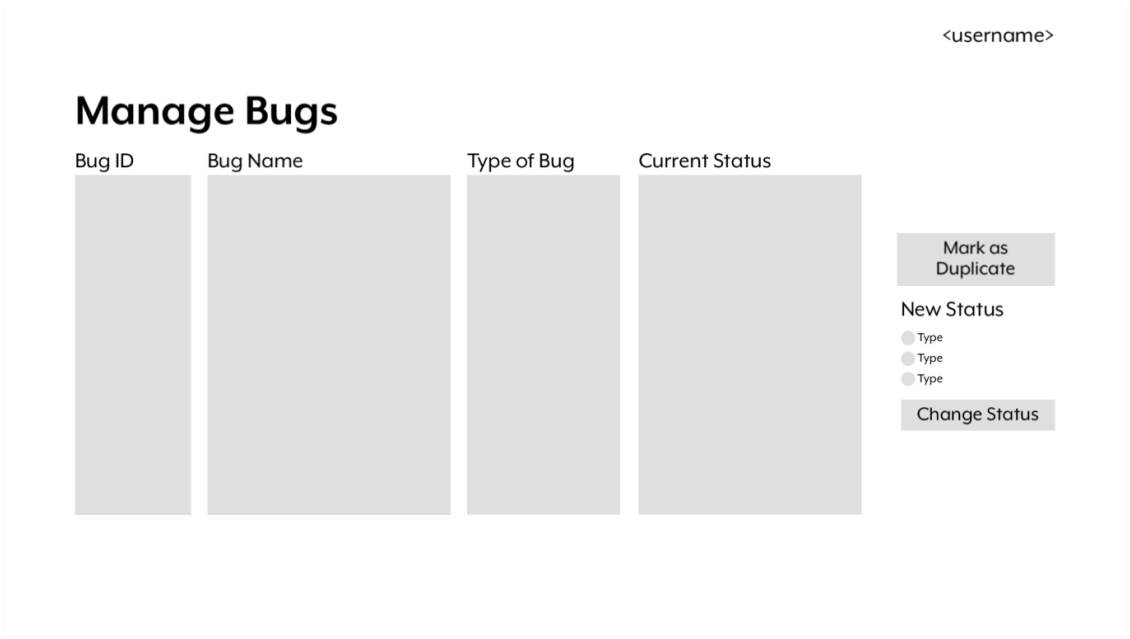

Figure 5.4: Manage Bugs page for a tester

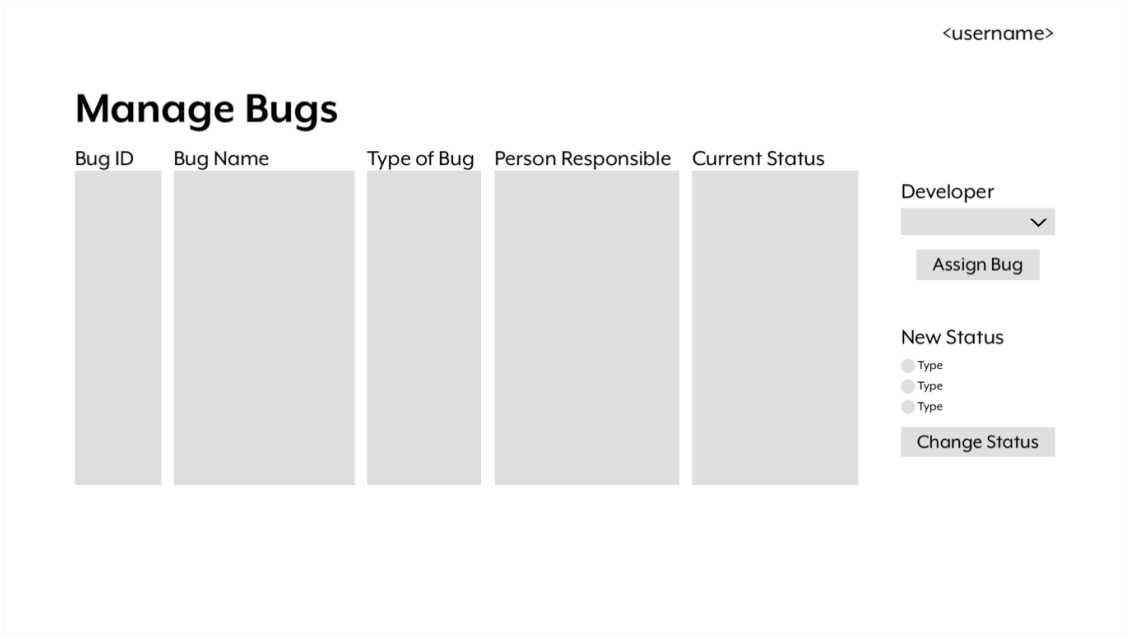

Figure 5.5: Manage Bugs page for a manager

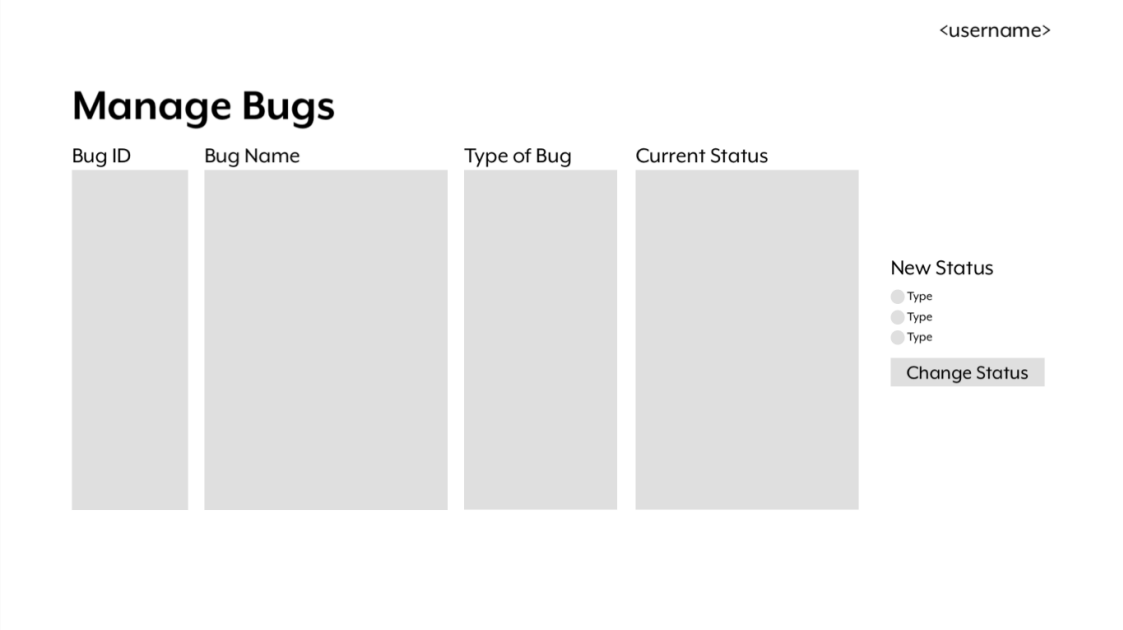

Figure 5.6: Manage Bugs page for a developer

Chapter 6

Technologies

6.1 HTML/CSS/JS

We will use Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS) for the front-end of the website. HTML describes the elements of the webpage using tags. CSS tells the client's web browser how to format those elements. JS is used for client-side scripting. The combination of these technologies lets us create an interactive website and customize it how we like. For example, to make a form, HTML would be used to describe the form and its fields, CSS would handle what it looks like, and JS would send data to the sever when the form is submitted.

6.2 Bootstrap

Bootstrap is a framework that integrates HTML, CSS, and JS used for web development that we will utilize to create user-friendly web pages.

6.3 Flask

Flask is a web microframework written in Python that we will use for the back-end of the web application, in other words, the software that the server will run. Flask will serve the clients the correct pages and handle their requests. All interaction with the database will be done through requests made to endpoints defined by Flask, which will call Python functions to handle them. It will also handle any other services like authentication.

6.4 MySQL

Our data will by stored in a MySQL database. MySQL is a relational database management system, which means it is composed of tables with rows and columns, and the entries of tables can have relations to other entries in other tables. We will use MySQL to store a list of all of the bugs with their data as well as account info.

Chapter 7

Architectural Diagram

7.1 Diagram

Figure 7.1 describes the architectural relationship between clients, server, and database.

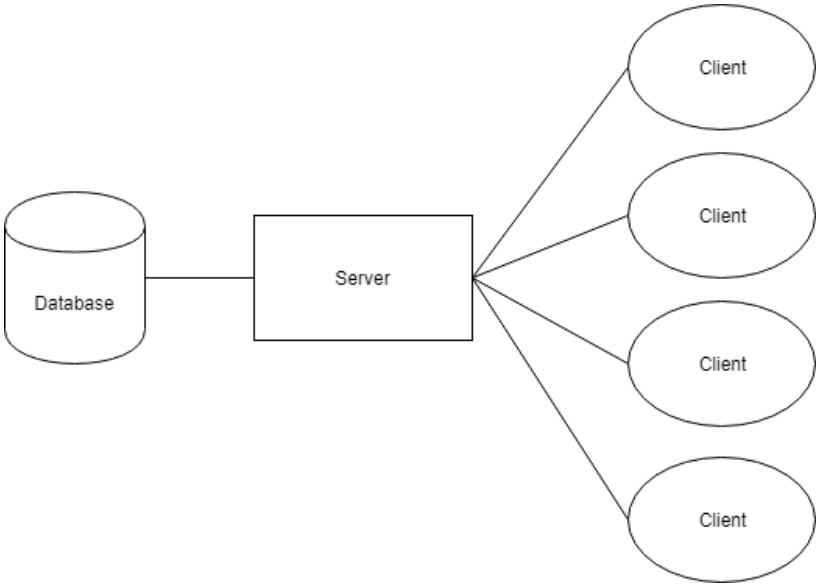

Figure 7.1: Architectural Diagram

7.2 Description

Our solution will use a client-server architecture with a database. This architecture is typical for web development and allows for easy processing of data between multiple clients and the server. The clients' web browsers handle part of the processing like handling user input and formatting the web page based on information given to them from the server. The server acts as an interface for the database. Clients send the server requests and the server will perform the appropriate operations on the database. The server also handles services like authentication and provides the HTML/CSS/JS for the web pages.

Chapter 8

Design Rationale

8.1 User Interface

Our web application is broken up into several web pages. Having different pages makes it easy to restrict access to certain types of accounts and simplifies the interface into understandable parts. Clients have access to a bug report page and a bug history page. These pages are separate so it is easy for new users to use the basic functionality of bug reporting without getting overwhelmed with the history page. Developers, testers, and managers all have pages to manage bugs. Although there is a large amount of overlap between these pages some elements are different so we gave each of the actors their own page. For example, the developer only should be able to change the status of a bug, while a manager should be able to assign a bug to a developer and change the status. These variations of the page are displayed in the rightmost column. Under the hood we can use templates to avoid copying too much code. All of the pages with a table of bugs also has a pop-up modal with detailed info of a specific bug. We chose to make this a pop-up instead of a separate page to minimize navigating between pages and make it easy to quickly look at bug information.

8.2 HTML/CSS/JS

Using these technologies is practically mandatory in web applications since it is what the browser needs to create web pages. There are frameworks that can compile other types of code to these technologies but we chose not to use any of them because our website is fairly simple and we want to keep the amount of technologies we need to learn to a minimum to save on development time.

8.3 Bootstrap

Bootstrap will be used as a library to establish a consistent looking interface for the user. This will benefit the user experience with our web application.

8.4 Flask

We chose to use Flask for our back-end framework because it is simple. Flask is a microframework which means it only contains the most necessary features to run a web application. That means the codebase we will need to learn about will be smaller than other frameworks. Our application does not require many extra features so we can afford to avoid larger frameworks. The only features we need that is not provided by Flask is database interaction and authentication, but Flask can be extended by Python libraries that provide those features. As a programming language Python is easy to learn which will help us learn how to use the framework quickly, and Python is available on the lab computers.

8.5 MySQL

Santa Clara University (SCU) provides a MySQL services that the lab computers can access, which makes this the easiest database solution for us to use. SQL databases are well documented, so it will be easy for us to work with. Other types of databases exist, but setting them up on the lab computers could be difficult and it is not worth it for us to learn about a new database technology.

Chapter 9

Test Plan

Below is a description of the various plans we will use to test the different aspects of the system.

9.1 Validation Testing

For validation testing, we will perform two types of tests: acceptance and usability testing. Acceptance testing is done when we bring a finished module or system to a user and have them run through it from start to finish in order to gauge if this is a product they would use our product when needed. Usability testing would be done in a similar manner, but as the user goes through the system we would have them make note on specific aspects they liked or didn't like, why, and what we might be able to do to improve their experience.

9.2 Verification Testing

For verification testing, we will perform three types of tests: unit, integration, and system testing. All of these will be done throughout development of the system. For unit testing, as we create each page, we will keep testing the functionality of each component. After testing a page individually, we will link it to the other developed parts of the site, and perform integration tests on them to see if they connect as intended. Once the whole system is put together, we will go through the four types of user flows and follow the paths to make sure each of the roles are functioning as intended.

Chapter 10

Risk Analysis

Table 10.1: Risk Analysis Table

Risk	Consequences	Probability	Severity	Impact	Mitigation
Time	- System isn't completed in time or not all aspects are finished	0.6	6	3.6	- Utilize lab time - Set deadline - Prioritize core features - Meet outside of class
Bugs	- System has errors, thus doesn't compile / work or has major vulnerabilities - Could affect the development timeline	0.99	3	2.97	- Keep code uniform - Comment a lot - Consistent, accurate documentation - Debug
Sickness	- Other team members have to take on more work than originally assigned - Decrease in quality of each aspect of the system due to overextension	0.2	6	1.2	- Take vitamins - Maintain good health habits - Work from bed
Loss of files	- Large time delay - Having to reconstruct whole project or parts of project from scratch or from memory	0.05	9	0.45	- Use collaborative online resources - Backup often - Ask team members before deleting files

Chapter 11

Development Timeline

The following table describes the development time of the project including who is in charge of what and when it will get done.

Table 11.1: Development Timeline

TASK TITLE	TASK OWNER	START DATE	DUE DATE	PCT OF TASK COMPLETE
Soft Deliverables				
Problem Statement	All	9/18/18	9/25/18	100%
Design Document	All	9/25/18	10/9/18	100%
Design Review Presentation	All	10/9/18	10/16/18	
Initial System Presentation	All	10/16/18	10/30/18	
Final Presentation	All	10/30/18	11/13/18	
Final Report + Demo	All	10/30/18	11/27/18	
Initial Operating System				
Front End Development	Sam	10/16/18	11/26/18	
Back End Development	Tom	10/16/18	11/26/18	
Test	Maddee	10/16/18	11/26/18	
Final Operating Systems				
Presentation Reflection	All	10/16/18	11/26/18	
Fix Design Errors	All	10/16/18	11/26/18	
Implement Recommended Requirements	All	10/16/18	11/26/18	
Final System Testing	All	10/16/18	11/26/18	
Final System Due	All	10/16/18	11/27/18	