

Final Year Project 2016/17



BugBot

A bot to fix bugs in code.

Sinead McDonagh 12337161
Electronic and Computer Engineering 4BP
National University of Ireland Galway
31st March 2017

Supervisor
Dr. Michael Madden
Senior Lecturer
National University of Ireland Galway.

CONTENTS

Table of Figures	6
Table of Tables	8
Acknowledgements.....	9
Abstract.....	10
1. Chapter 1 Introduction	11
1.1 Project Motivation	11
1.2 Project Objectives	12
1.3 Project Outline	14
1.4 Overview of BugBot	14
1.5 Structure of Thesis	16
2. Chapter 2 Technology Review.....	17
2.1 Open Source Code	17
2.2 Distributed Version Control Systems	17
2.2.1 <i>Git Functionality</i>	19
2.2.2 <i>GitHub & BitBucket</i>	21
2.3 Static & Dynamic Code Analysis	22
2.4 Error Types	23
2.4.1 <i>Syntax Errors</i>	23
2.4.2 <i>Semantic Errors</i>	23
2.4.3 <i>Compile Time Errors</i>	25
2.4.4 <i>Run Time Errors</i>	25

2.5	Existing Technologies.....	26
2.5.1	<i>Integrated Development Environments</i>	26
2.5.2	<i>Compilers</i>	26
2.5.3	<i>FindBugs</i>	27
2.5.4	<i>Commercial Static Analysis Companies</i>	28
2.5.5	<i>Coverity Scan</i>	28
2.5.6	<i>Lint</i>	30
2.5.7	<i>LAVA</i>	30
2.5.8	<i>Summary of Static Analysis Tools</i>	31
2.6	Technologies Used.....	32
2.6.1	<i>Java Project Using Maven</i>	32
2.6.2	<i>Database Technology</i>	32
2.6.3	<i>Online Code Repositories</i>	33
2.6.4	<i>Web Crawler</i>	33
2.6.5	<i>Visual Studio and Visual Basic</i>	34
2.6.6	<i>Website Creation</i>	34
3.	Chapter 3 Design & Implementation	35
3.1	Requirements	35
3.1.1	<i>Design Requirements</i>	35
3.1.2	<i>Technical Requirements</i>	36
3.2	System Components.....	38
3.3	Backend Code	39
3.3.1	<i>Top Level Code Flow</i>	39

3.3.2	<i>Classes & Corresponding Key Functions</i>	40
3.3.3	<i>Code Flow Charts by Class</i>	47
3.3.4	<i>Backend Code Analysis</i>	49
3.3.5	<i>Variable Identifying & Analysis</i>	53
3.3.6	<i>Bug Identifying & Fixing</i>	55
3.3.7	<i>Databases Queue & Record</i>	57
3.3.8	<i>Web Crawling</i>	59
3.4	<i>BugHive Database</i>	60
3.4.1	<i>Bug Definitions</i>	60
3.4.2	<i>Bug Languages</i>	64
3.4.3	<i>List of Bugs</i>	66
3.5	<i>BugBot Navigation</i>	73
3.5.1	<i>Autonomous Navigation: Web Crawling BitBucket</i>	73
3.5.2	<i>Direct Request Users: Requested Navigation</i>	73
3.6	<i>Administrator GUI</i>	75
3.6.1	<i>Visual Basic using Visual Studio Implementation</i>	75
3.6.2	<i>Administrator's Role</i>	77
3.6.3	<i>Main Panel</i>	77
3.6.4	<i>BugHive Panel</i>	78
3.6.5	<i>Graphs Panel</i>	79
3.7	<i>User Website & Twitter Account</i>	80
3.7.1	<i>User Information</i>	81
3.7.2	<i>Bug Addition</i>	82

3.7.3	<i>Direct Requests</i>	82
4.	Chapter 4 Results	89
4.1	Testing.....	89
4.1.1	<i>Testing Regular Expression Patterns</i>	89
4.1.2	<i>BugBot Probation</i>	91
4.2	BugBot Data Recording	94
4.2.1	<i>BugBotReport.txt</i>	94
4.2.2	<i>DATA.csv</i>	94
4.2.3	<i>BugBotConfig.txt</i>	95
4.3	Performance and Statistics	96
4.3.1	<i>Successful Identification & Fix Case Study</i>	96
4.3.2	<i>Repository Scanning & Bug Finding Statistics</i>	99
4.3.3	<i>Discussion of Findings</i>	105
5.	Chapter 5 Conclusions	106
5.1	Challenges.....	106
5.1.1	<i>Unauthorised Bot Access</i>	106
5.1.2	<i>Bug Gathering</i>	106
5.1.3	<i>Creating a Presence</i>	107
5.2	Issues & Limitations	108
5.2.1	<i>Bot Limitations</i>	108
5.2.2	<i>Limitations of Bug Identification</i>	108
5.3	Future of BugBot	110
5.3.1	<i>Future Bug Additions & Language Capabilities</i>	110

5.3.2	<i>Integration of FindBugs</i>	112
5.3.3	<i>Cloud Based Deployment</i>	113
5.4	Achievements.....	114
	References.....	115
	Appendices.....	121

TABLE OF FIGURES

FIGURE 1 GOOGLE TRENDS GIT, MERCURIAL & SUBVERSION [22].....	18
FIGURE 2 VISUAL OF GIT FUNCTIONS [23].....	19
FIGURE 3 GOOGLE TRENDS GITHUB VS BITBUCKET [22].....	21
FIGURE 4 GOOGLE TRENDS COMPARISON OF GIT CAPABLE REPOSITORY HOSTING SITES [22]	21
FIGURE 5 SYNTAX ERROR DETECTED BY COMPILER EXAMPLE.....	23
FIGURE 6 SEMANTIC ERROR EXAMPLE	24
FIGURE 7 SEMANTIC ERROR UNDETECTED BY COMPILER EXAMPLE.....	24
FIGURE 8 POTENTIAL RUN TIME ERROR VS COMPILE TIME ERROR.....	25
FIGURE 9 TOP LEVEL SYSTEM INTERACTION	38
FIGURE 10 TOP LEVEL CODE FLOW CHART	39
FIGURE 11 DRIVER CODE BEFORE CREATING A REPOSCAN OBJECT	40
FIGURE 12 ACCESSOR METHOD GETFIRST() OF QUEUE DATABASE.....	43
FIGURE 13 GETREGEX() METHOD OF THE BUGHIVE CLASS	44
FIGURE 14 TWEET CREATED FROM THE ADMIN CLASS.....	46
FIGURE 15 BUGBOT TWITTER FEED.....	46
FIGURE 16 KIVIAT METRICS GRAPH AT BASELINE	49
FIGURE 17 BASELINE METRIC GRAPH	50
FIGURE 18 REDUCING COMPLEXITY OVER 15 CHECKPOINTS	51
FIGURE 19 POST REVIEW & EDIT KIVIAT AND COMPLEXITY DISTRIBUTION GRAPHS	51
FIGURE 20 DECLARED AND UNDECLARED VARIABLES.....	53
FIGURE 21 METHOD TO RECORD UNDECLARED VARIABLES.....	54
FIGURE 22 METHOD TO RECORD DECLARED VARIABLES.....	54
FIGURE 23 BUGSCAN FLOW CHART	57
FIGURE 24 QUEUE DATABASE AS OF 28TH MARCH 2017	58
FIGURE 25 RECORDS DATABASE AS OF 28TH MARCH 2017	58
FIGURE 26 FOR BITBUCKET WEB CRAWLING	59
FIGURE 27 ATTAINING AN AUTHORISATION TOKEN ON GITHUB	59

FIGURE 28 MOST USED REGULAR EXPRESSION SYMBOLS FOR BUG DEFINITIONS	63
FIGURE 29 IEEE GRAPH OF PROGRAMMING LANGUAGE POPULARITY [54].	64
FIGURE 30 THE MOST POPULAR PROGRAMMING LANGUAGES ON GITHUB [55]	65
FIGURE 31 BUGHIVE AS OF 14TH MARCH 2017	66
FIGURE 32 MAIN PANEL OF ADMINISTRATORS GUI FLOWCHART.....	75
FIGURE 33 VISUAL BASIC CODE FOR CONNECTING TO BUGHIVE DATABASE.....	76
FIGURE 34 VISUAL BASIC CODE TO DELETE A BUG FROM BUGHIVE	76
FIGURE 35 ADMINISTRATOR GUI- MAIN PANEL.....	77
FIGURE 36 ADMINISTRATOR GUI- ACCESS TO BUGHIVE	78
FIGURE 37 ADMINISTRATOR GUI GRAPH - BUG FREQUENCY	79
FIGURE 38 ADMINISTRATOR GUI GRAPH - LANGUAGE DISTRIBUTION	79
FIGURE 39 BUGBOT.ORG FIRST PAGE TOP PANEL.....	80
FIGURE 40 BUGBOT TWEET- 1000TH REPOSITORY ACCESSED	81
FIGURE 41 BUGBOT TWEET- 12TH BUG ADDED TO BUGHIVE.....	81
FIGURE 42 BUGBOT.ORG/BUGHIVE SLIDESHOW	82
FIGURE 43 WEBPAGE WWW.BUGBOT.ORG/HOME	84
FIGURE 44 WEBPAGE WWW.BUGBOT.ORG/ABOUT	85
FIGURE 45 WEBPAGE WWW.BUGBOT.ORG/BUGBOT	86
FIGURE 46 WEBPAGE WWW.BUGBOT.ORG/BUGBOT	87
FIGURE 47 WEBPAGE WWW.BUGBOT.ORG/BUGHIVE	88
FIGURE 48 REGEX TESTING SOURCE CODE.....	90
FIGURE 49 CONSOLE OUTPUT OF REGEX TEST PROGRAM.....	90
FIGURE 50 BUGBOT'S INCORRECT IDENTIFICATION OF A BUG.....	91
FIGURE 51 BUGBOT INCORRECT IDENTIFICATION OF A BUG	92
FIGURE 52 BUGBOT INCORRECT FIX OF A BUG	93
FIGURE 53 BUGBOT REPORT AS COMMITTED TO A REPOSITORY	94
FIGURE 54 DATA.CSV FILE	95
FIGURE 55 BUGBOTCONFIG DEFAULT VALUES.....	95
FIGURE 56 THE PRINT TO CONSOLE FOLLOWING 8 BUG FIXES.....	97

FIGURE 57 EXAMPLE "DIFFS" AFTER BUGBOT FIXING.....	97
FIGURE 58 THE COMMIT OF THE BUGBOTREPORT.TXT FILE	98
FIGURE 59 THE CURRENT NETWORK GRAPH.....	98
FIGURE 60 LANGUAGE DISTRIBUTION AMONG CLONED BITBUCKET REPOSITORIES	99
FIGURE 61 FILES CLONED & NOT SCANNED.....	99
FIGURE 62 TIME TO SCAN VS LINES OF CODE	100
FIGURE 63 LINES OF CODE VS FILE SIZE.....	101
FIGURE 64 NUMBER OF FILES VS BUGS FOUND	101
FIGURE 65 BUG DISTRIBUTION.....	102
FIGURE 66 BUGS FOUND BY LANGUAGE.....	103
FIGURE 67 PROJECT SUBMISSION ON GITHUB.COM/SINEADMCD/BUGBOT	121
FIGURE 68 ADMIN GUI SUBMISSION ON GITHUB/SINEADMCD/GITHUB	121

TABLE OF TABLES

TABLE 1 COMPARISON OF STATIC ANALYSIS TOOLS	31
TABLE 2 VISUAL OF VARIABLE ARRAY STRUCTURE	53
TABLE 3 REGULAR EXPRESSION TO IDENTIFY VARIABLE TYPES.....	54
TABLE 4 EXAMPLE OF VARIABLE ARRAY STRUCTURE	55
TABLE 5 AVERAGE FIGURES OF BUGGY REPOSITORIES.....	102
TABLE 6 REPOSITORIES REPEATING THE SAME BUG	104
TABLE 7 DIRECT REQUEST WAIT TIMES.....	104
TABLE 8 BUGBOT'S PROGRESS	114

ACKNOWLEDGEMENTS

I would first like to thank my supervisor, Michael Madden for his continued guidance and support throughout the development of this project. For the immense amount of time he invested in offering valuable comments and encouraging feedback from the beginning to end, without which the project would be of no comparable standard.

I would also like to thank my peers in the Engineering class of 2017 for their welcomed criticism and willingness to help with testing as well as providing an open forum for complaint.

Finally, I would like to express my sincerest gratitude to my parents who were given no choice but to proof read this report despite having no knowledge or interest in the topic.

ABSTRACT

In 1996, the European Space Agency launched the Ariane 5, a rocket intended to send expensive satellites into the Earth's orbit and simultaneously ensure Europe's presence in the international space industry. All of 10 years and \$7billion was invested in the Ariane 5, but the system shut down a mere 36 seconds after leaving the ground due to one bug. The software attempted to load a 64-bit integer into a 16-bit register causing an overflow and triggering a system self-destruction just two and a half miles from the ground [1].

CHAPTER 1 INTRODUCTION

1.1 PROJECT MOTIVATION

Reliability is one of the most important attributes of software quality, and bugs are detrimental to its reliability. The occurrence of bugs in code can be minimized but never truly eliminated while programmers remain human. So, for now, debugging code remains an unavoidable reality and integral to the development of software.

As the software development market expands year-on-year with increased complexity of products, shortened development cycles, and higher customer expectations, software bugs cause an undesirable increase in the cost of development both in terms of time and revenue. In 2002 Hailpern and Santhanam stated that software debugging, testing and validation claims 50% - 75% of software development costs [2]. The same year, the National Institute of Standards and Technology, NIST reported that software defects cost the American economy \$60 billion annually [3]. In response to these staggering figures, many companies have been set up to exploit the issue, the services they offer and corresponding fees they charge are further discussed in section 2.5.

I believe that much like the concept of open source code de-capitalising and accelerating the programming industry, so too should the debugging of this code. On my first introduction to the ideological world of open source code, I was somewhat confused as to why so many highly skilled developers were contributing to free products and services. To understand the concept, I re-evaluated my view on what software development is. It seemed that much like scientific research and advancements being public information, to an extent, for the betterment of humankind and the acceleration of acquiring knowledge in a certain field, developers were adopting the same strategy [4]. Advancements can be made faster in software algorithms and techniques when information is shared; in a 2013 study carried out by static analysis company Coverity Scan, it was reported that the quality of open source code was significantly higher than that of proprietary code scoring defect densities¹ of

¹ Defect density is defined as 1.0 for 1 defect per every 1,000 lines of code.

0.59 and 0.72 respectively [5]. Even so, with the masses of open source code available on online code hosting sites free for viewing and use by the public, it seemed natural then that there should exist automated tools to service this code, that is, debug it.

1.2 PROJECT OBJECTIVES

The aim of this project is to provide programmers with a bug scanning tool that requires little to no time, resources and effort to find and fix the bugs in their code. To reduce the time spent by developers debugging code by automating the process through a commonly used tool – static code analysis. By incorporating widely used open-source code hosting websites, this project aims to implement this tool in a new way.

Static code analysis tools generally require a developer to install a plug-in to their IDE or to sign up and pay an expensive subscription fee to companies for code analysis, this is further discussed in section 2.5.4. These services, although providing viable bug-scanning services, require time and resources. Static code analysis tools search code for bugs that remain undetected by compilers and do not present themselves through run-time errors due to their syntactical correctness, these are referred to as semantic errors [6]. A program with semantic errors can execute without any errors being reported but can often lead to unexpected and incorrect returns from a program, and the programmer may never be made aware of the issue that is either knowingly or unknowingly returning incorrect results. These are the type of bugs this project aims to identify and fix. To detect semantic errors, the code does not need to be run as they present themselves in context and result in unexpected outputs that can be difficult to trace back from. Static code analysis is further discussed and compared with dynamic code analysis in section 2.3.

This project aims to scan for these bugs remotely. The program that is bug-scanning developers' code remains detached from the developer as no download or installation is necessary. The only effort required from a user requesting the service is to provide the remote location of their code. This implies no compatibility problems that have been issued in the past with static code analysis plug-in products such as FindBugs [7].

In addition to helping developers debug their code when requested, this service also aims to eliminate the defined bugs from all code available on remote code hosting websites. By autonomously scanning code in remote repositories; the service strives for an overall improvement in software quality, and reliability by proactively seeking out bugs in online code repositories. This saves programmers time completely, as no other effort is required, except simply merging any fixes with the original code.

Furthermore, the service intends to offer all developers a platform to edit, add and raise issues with bugs scanned for, much like the open source software movement encourages. This should offer a level of transparency as well as accelerate the expansion of the project.

As there are an endless number of bug possibilities to be defined, the project aims to be expandable; to cover a broader range of programming languages and a broader range of bugs. For a truly successful and helpful tool, there should be no end to its expansion. Thus, the program is written with the goal of being dynamic and allowing for additions of bugs with minimal change to the source code.

Due to the malicious use of bots on code repositories in the past; changing functioning code to endorse different classes and frameworks, this project aims to offer a user-friendly and accommodating approach to static code analysis and automated fixing. To do this, it is imperative that the identification and fix of a bug be correct, it must avoid false positives. The aim is to only improve developers code and not irritate them with useless and impractical “fixes.” For this reason, an extensive amount of testing and validation on each implemented bug is required.

Although numerous static code analysis tools are available to users, none fulfill the objectives outlined above in their entirety. The existing technologies differ regarding the formal methods they implement and the programming languages they support, this is summarised in section 2.5.8. This project aims to be a newer implementation of static code analysis exclusively developed for remote repositories.

1.3 PROJECT OUTLINE

To fulfill the above objectives, this project comprises an autonomous web-crawling bot that finds and clones online git repositories in search of bugs. It implements static code analysis on each line of code in the repository that is written in a supported language; Java, C, and C++.

The program uses regular expression and pattern matching techniques to identify a bug from a defined list. Predefining these bugs involved collecting a solid set of candidate bugs and analysing every possible syntax, mechanism, and output of the bug in a variation of programming languages. They are defined to include the relevant details to identify, explain and fix the bug. Once a bug is identified the bot will respectively edit the code to eliminate the bug.

As this project is to deal with online public code repositories, specifically git code repositories, the web-crawling bot carries out git functionality necessary to successfully contribute to a repository. This git functionality is further discussed in section 2.2.1.

In addition to its autonomous navigation through repositories, it is also equipped to handle direct requests from developers by supplying the URL of their online repository or by the inclusion of a FindMe file in their repositories.

1.4 OVERVIEW OF BUGBOT

BugBot is a bot that interacts with remote code repository sites such as GitHub [8] or Bitbucket [9]. It forks and clones' other developers open-source programs and implements static code analysis on each line of the program to find lines that contain bugs. It fixes the bug, commits and pushes the change back to the forked repository and makes a pull request along with a helpful message.

The bugs defined for BugBot are stored in a MySQL [10] database referred to as the bug-archive or BugHive [11]. The BugHive allows for the easy addition and maintenance of the bugs by an administrator and outlines all the necessary criteria for BugBot to identify, fix and explain the defined bugs. BugBot's website also includes an online read-access version of the BugHive, with the current,

implemented version of the bug database available for user reading. It also includes functionality for users to make bug suggestions that could be added to the BugHive.

BugBot is optimised for three specific use cases; the administrator, autonomous unrequested bug-scanning of remote repositories and users that directly request bug-scanning of their remote repositories.

For an administrator, a simple front-end system allows for the maintenance of the BugHive as well as current performance statistics, current bot status, an option to reset the bot and a live feed of BugBot's Twitter [12].

The main use cases concern the bot's navigation through code repositories. BugBot autonomously navigates through code repositories until a direct request to a repository is made. These direct requests are made either through BugBot's website [13] or by adding a BugBotFINDME.txt file to an online repository. Both the autonomous jobs and direct request jobs are organised in a first-in-first-out queue for bot attention along with a timestamp of when the request was made. Once scanned, the repository URL's are recorded to ensure BugBot does not repeatedly scan the same repository, the only exception for repeat scans is when directly requested to do so through the website. This functionality is further described in section 3.5.2.

BugBot manages a bot-controlled Twitter account to update users/followers on things like new bugs added to the bug archive, periodical statistic reports and live progress of the bot's navigation and bug finding/fixing.

1.5 STRUCTURE OF THESIS

Chapter 1 Introduction: This chapter highlights the need for an automated static code analyser for remote repositories and a method of fulfilling this. In addition, it outlines the objectives of this project and a summary of functionality provided by BugBot.

Chapter 2 Technology Review: This chapter describes the technology that BugBot depends on and incorporates such as distributed version control, static code analysis, and open source code. It also analyses existing tools that could be compared to BugBot and outlines the technologies used to develop BugBot such as Java and databases.

Chapter 3 Design and Implementation: This chapter outlines the design and technical requirements of BugBot and a detailed walk-through of how the requirements were met through various system components and overall system interaction.

Chapter 4 Results: This chapter describes the testing mechanisms implemented to validate BugBot as a tool and presents the results gathered on bug finding, time taken and statistics of scanned repositories. Furthermore, this chapter presents a case study of successful bug finding and fixing.

Chapter 5 Conclusion: This chapter outlines the challenges overcome through the development of this project, the issues, and limitations of using bots to fix human written code and the future of BugBot regarding bug additions and language capabilities.

CHAPTER 2 TECHNOLOGY REVIEW

2.1 OPEN SOURCE CODE

The use of online code repository sites has increased considerably in recent years in correspondence with the open source software movement. The open source movement was based on the premise that sharing code publicly would accelerate the advances made in software with freely distributed code and providing developers with open-access to application's source code [14].

Linux is an open source computer operating system founded by Linus Torvalds and is a well-known open source success story. For a long time, Linux was not using any version control system, and additions and changes to the source code were passed around using patches and archived files. In 2002 they began using BitKeeper [15] to manage the source code of the Linux Kernel [16]. However, in 2005 when Linux's relationship with BitKeeper broke down the developers needed to create something similar to BitKeeper. Thus, the creation of git, a distributed version control system.

2.2 DISTRIBUTED VERSION CONTROL SYSTEMS

Version control systems are a suite of software tools that allow software developers to track and manage changes by numerous contributors to source code over time. It saves every state of the code as it is being developed allowing easy roll-back if an issue is encountered [17]. As of 2008, git surpassed the popular version control system; Subversion, and became the largely preferred source code management system.

Subversion [18] is a centralised version control system. It is an open source system that facilitates distributed file sharing, but only one central server hosts an entire project including all its previous

states. When a contributor “checks out”² a project to work on they receive a limited view of the project on their local machines [19].

Git [20] and Mercurial [21] are distributed version control systems. They offer developers everything Subversion offers with the addition of offline source control and speed. Distributed means that everyone working on a project has their own working repository i.e. the entire project including all history and not just a version of it. It allows non-linear development of code while all client changes to source code remain local until synchronised to the centralised repository. This offers the speed that Subversion did not provide

Although Git and Mercurial are both distributed version control systems, this project focuses on Git repositories as it is the widely preferred distributed version control system of choice among developers. This is evident in the Google Trends figure 1. This graph also reflects the dying interest in centralised version control system Subversion which reigned superior up to 2008 according to this graph.

This graph was created on Google Trends [22] by the author on the 16th March 2017. Furthermore, all Google Trends graphs displayed in this report were created by the author.



Figure 1 Google Trends Git, Mercurial & Subversion [22].

Numbers represent search interest relative to the highest point on the chart at a given time. A value of 100 is the peak popularity for the term.

² Check out is a subversion equivalent of cloning.

2.2.1 Git Functionality

The main functions involved in successfully contributing to a git repository³ include forking, cloning, committing, pushing and making a pull request. Each is an entirely different function but are all closely linked in fulfilling the aim of contributing code in a git repository.

Figure 2 [23] is an example of a general git network graph. The top line is the master branch or trunk labelled with different versions; V1.0, V2.0, and V3.0. It is evident that in this example that no commits are directly made to the master branch or trunk but instead the trunk has been forked, and feature branches have been made and committed to before being merged to the trunk and a new version released. The various functions labelled in this figure are described in further detail below.

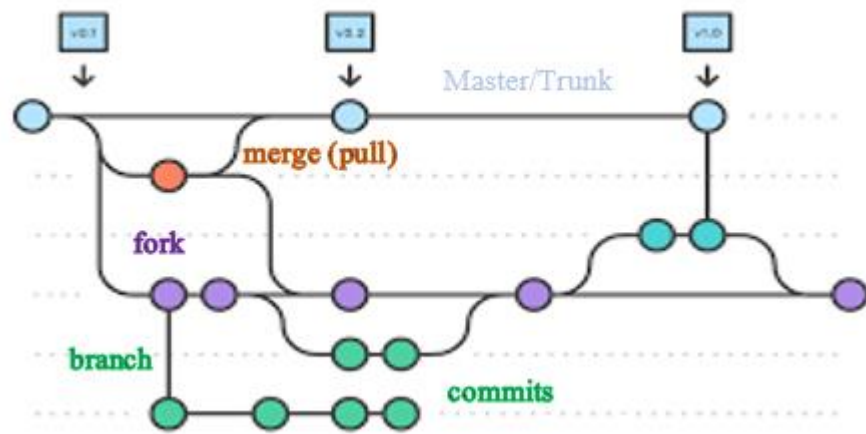


Figure 2 Visual of Git Functions [23]

2.2.1.1 Master/ Trunk and branches

The main working project is the master or trunk of the repository. As a general rule, this should always remain buildable, and no extensive changes should be made to the trunk. Instead, changes should be made to branches which are essentially “save as” commands for repositories. Branches are working copies of the trunk, to which changes should be made and features should be added via commits. These additions should then be tested before making a pull request to merge with the trunk.

³ A git repository is a repository using the git distributed version control system.

2.2.1.2 Forking

Forking a developer's remote repository remotely copies the trunk or master branch and distributes a working copy to the user in the form of a new repository. The user can now work on their forked repository without making changes to the original.

2.2.1.3 Cloning

A user can work online or offline with git. Cloning a git repository to a local device allows offline editing of a local repository that is ultimately linked to the remote repository. Working locally offers the speed and efficiency that subversion did not provide as developers can carry out commits, diffs, branches, and merges without being connected to a network.

2.2.1.4 Committing

Users commit their code after making changes to the code. Generally, commits should be made frequently after a few changes all contributing to one function or issue resolve. Commits can be made locally and a history of the commits are saved locally until synchronised to the remote repository.

2.2.1.5 Pushing

After committing locally or remotely once or multiple times, the commits can be pushed to the remote repository where all commit history is now saved remotely. The repository is pushed to the branch it was initially cloned from or to the master if no branch was made.

2.2.1.6 Pull Requests

Making a pull request on Git is making a request to merge a commit to another branch or to the trunk. Git can seamlessly merge repositories due to its parallel working capabilities and the repositories are never refused for being "out-of-date". This was a common issue with subversion as all changes were made separately and once the central or trunk repository was merged with once, all other working copies were now declared out-of-date.

2.2.2 GitHub & BitBucket

With the increase in git source code management- respectively arose an increase in the use of git capable online code repository sites. The two most popular of which are GitHub and BitBucket, evident in figure 4. These sites require users to sign up to receive remote repository hosting, free of charge with the exception of GitHub's monthly charge for private remote repositories.

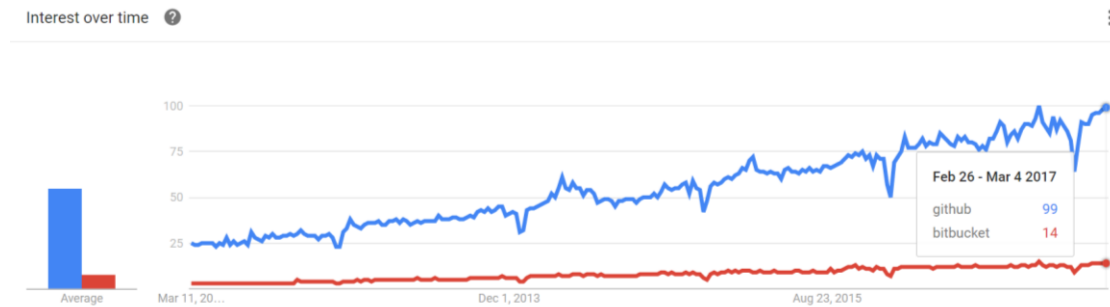


Figure 3 Google Trends GitHub vs BitBucket [22]

GitHub is the most popular distributed version control and source code management site currently used by developers. It boasts 20 million users and over 55 million hosted projects [24]. BitBucket is another git capable source code management site with over 5 million users as of July 2016, GitHub and Bitbucket are the two most popular distributed version control sites.

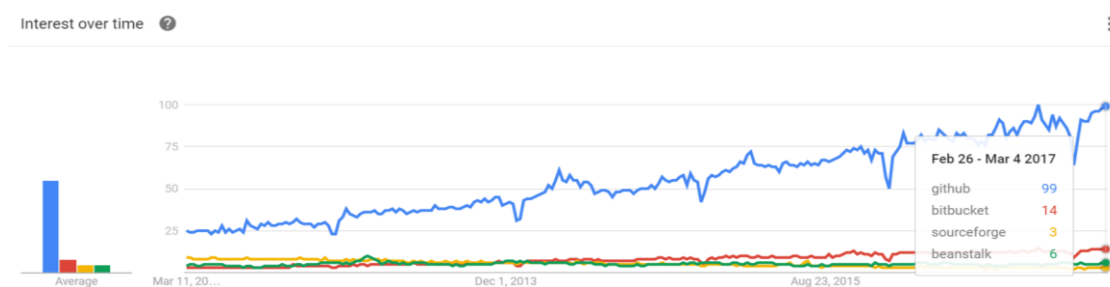


Figure 4 Google Trends comparison of Git capable repository hosting sites [22]

2.3 STATIC & DYNAMIC CODE ANALYSIS

There are two main types of code analysis; static and dynamic. Static code analysis operates by scanning code for bugs without executing it, whereas dynamic code analysis executes the code on a virtual processor and finds bugs based on insufficient test inputs or untimely behaviour [25].

Static code analysis is the analysis of code without running or building the code. It deals with the observation of code in its syntax and not in its mechanism and draws conclusions on the program behaviour due to a semantic error or syntax, different types of errors are discussed in section 2.4. In 1990, the IEEE defined static code analysis as "The process of evaluating a system or component based on its form, structure, content, or documentation" [26]. The identification of bugs in this way can both be easier and more complex as some bugs can be obvious and easily detected and others can be hidden among many variable references [27].

Dynamic code analysis is the process of scanning for bugs by running the code and monitoring its undesirable outputs and mechanisms. This requires a method for running the code and a specific outline of what constitutes failure or the presence of bugs. Dynamic code analysis is a timely process as often times its testing requires users to create test cases that might constitute failure [28].

The advantages of automated static code analysis include its ability to point out weaknesses in the code at an exact location, the speed of an automated tool, its ability to scan an entire code base and capability to point out weaknesses early on in the software development lifecycle as well as in the finished product. On the contrary, static analysis has many limitations. It is much more susceptible to inducing false positives or false negatives as it does not analyse the mechanism of the program itself, it is limited to the programming languages it is defined to support, and furthermore, the rules it is scanning with.

2.4 ERROR TYPES

Below I discuss the terms used to describe different programming errors, examples of each and the tools used to detect these errors. The tools mentioned are then further discussed in section 2.5.

Although each error is discussed individually the errors are not decoupled from each other; a syntax error is also a compile time error and semantic errors can introduce themselves at compile time and run time.

2.4.1 Syntax Errors

Syntax errors can include unmatched brackets, incorrect spelling of variable declarations and the incorrect use of upper or lower case letters. Some syntax errors are unique to a programming language and some are common among many programming languages. Various IDE's text editors can identify syntax errors, for example NetBeans' text editor can identify and fix unmatched brackets, however even if the text editor does not detect these syntax errors, they are identified by the compiler as it tries to translate the source code.

```
int first = 5;  
System.out.println("my first integer: " + First);
```

Figure 5 Syntax Error Detected by Compiler Example

Figure 5 is an example of a syntax error and compile time error. "First" is spelled with an upper-case letter, this is essentially a misspelling. As the compiler attempts to translate this source code, it finds a variable it is unable to translate as it has not been declared. These errors are quickly identified and demand the attention to be fixed and so BugBot does not define these errors as bugs as it is assumed they would not be present.

2.4.2 Semantic Errors

Semantic errors are logical errors whereby the code written is syntactically correct but produces incorrect or unintended results. The program can often execute without reporting any errors but a

syntactically valid structure of statement does not infer no error is present. Some compilers can identify certain types of semantic errors but other more implicit semantic errors go undetected and can be difficult for a developer to identify [29]. These are the types of errors BugBot is concerned with.

An example of a semantic error identified by the NetBeans compiler is calling a method with a return statement without assigning the returned variable or assigning it to the wrong variable type. The compiler will point these errors out to the user and ask to be fixed.

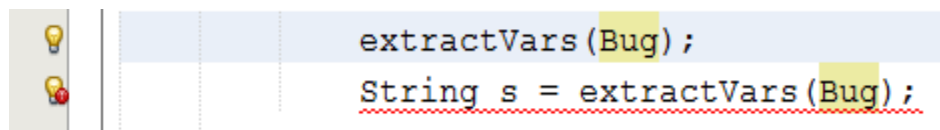


Figure 6 Semantic Error Example

In the example outlined in figure 6, `extractVars` is a method that takes a string parameter and returns a List of Strings. In the first line of code a compiler “hint” appears in the margin which warns us that the return of the method has not been assigned. In the second line, the compiler more seriously warns us that the returned variable is incompatible with the assignment. BugBot chooses not to deal with these errors that are clearly pointed out by compilers.

An example of a semantic error that is not pointed out by a compiler is shown below in figure 7.

```
double d = Double.NaN;
if (d==Double.NaN)
{
    // this code is never reached
}
```

Figure 7 Semantic Error Undetected by Compiler Example

The error shown in figure 7 is syntactically correct but logically meaningless as no double variable is ever equal to `Double.NaN`, not even `Double.NaN` for that matter. The compiler does not point out any warnings or errors with this statement despite the code never being reached. This is the type of error BugBot is concerned with.

2.4.3 Compile Time Errors

Compile time errors are identified by the compiler when attempting to translate the code from the interpretive language to machine language. These are static and can be syntax errors or semantic errors as presented above. If a compile time error has not been fixed before running the code it will result in an error when the statement is reached, this does not however mean it is recategorized as a run-time error. Compilers are further discussed in section 2.5.2.

2.4.4 Run Time Errors

Run time errors are errors that occur when the code is being executed. Dereferencing a null pointer or a division by zero are examples of run-time errors, they will effectively be pointed out to the programmer during the execution of the program. These are dynamic and debugging tools such as break-points facilitate developers in identifying these types of bugs in addition to the creation of unit tests.

Figure 8 illustrates clearly the difference between compile time and run time errors. Immediately the compiler has pointed out that the String variable “my_value” cannot be converted to type int. On the contrary it does not identify a problem with parsing the value to an integer as this is syntactically and logically correct; given a certain condition. If, for instance, the value read from the console happens to be a word “abc” which is impossible to parse as an integer, this program will result in a run time error. However, if the value read from the console is an integer value “45” the program will succeed. This is dynamic as some test cases may succeed and others may fail.

```
String my_value = System.console().readLine();  
  
int c = my_value; // compile time error  
  
int r = Integer.parseInt(my_value); // run time error
```

Figure 8 Potential Run Time Error vs Compile Time Error

2.5 EXISTING TECHNOLOGIES

Below, I discuss the various technologies and products that exist to find and/or correct bugs in code and a means of testing these technologies. Although Integrated Development Environments and compilers are discussed it is to outline the bugs BugBot is unconcerned with.

2.5.1 Integrated Development Environments

Integrated Development Environments or IDEs are software applications to facilitate programmer's software development. They provide text editors tailored to programming languages, build automation and debugging tools and integrated compilers. IDEs text editors can point out errors such as basic syntax errors and unmatched brackets. IDEs such as NetBeans [30] include debugging tools such as the ability to add break-points to source code, add field watches, and step through code. These are useful to detect dynamic errors and some can point out what line the error is present. In addition to this, various IDEs also allow users to install additional static analysis plugins for further debugging capabilities such as FindBugs for NetBeans and Eclipse, discussed below.

2.5.2 Compilers

A compilers purpose is to translate an interpretive language such as Java, Python and Ruby into machine language. It points out what are called compile time errors, discussed in the previous section, before the IDE runs the code. These can be the consequences of a typing slip or a case of the programmer not completely understanding the implemented code. These are non-technical and easily detected, and many compilers offer suggestions to "fix" the error.

This project is unconcerned with compile time errors or even run time errors as they are already clearly exposed to programmers and demand the attention necessary to be fixed.

BugBot will be more concerned with the bugs that go undetected by compilers due to their correct syntax and functional declaration i.e. semantic errors as discussed in section 2.4.

2.5.3 FindBugs

FindBugs [31] is a free Eclipse and NetBeans plug-in developed specifically for the Java programming language that implements static code analysis. It's an open source project on SourceForge.net developed by the University of Maryland and operates under the Lesser General Public Licence or LGPL.

FindBugs reported 270,000 downloads world-wide as of 2006, with users from countries such as the USA, Germany, China and Japan and is widely regarded as one of the most popular open-source static analysis tools.

FindBugs works by analysing Java bytecode which are the compiled class files for an open access bug collection containing 424 bugs that was last amended in March 2015.

The types of bugs FindBugs looks for are outlined as:

- System-specific logical bugs (e.g. logical errors in algorithms, calling wrong API methods in wrong places, incorrect checks for permissions, wrong data used in wrong places)
- User interface bugs (e.g. JavaScript errors, layout issues)
- Server configuration bugs (e.g. issues with character encodings, problems caused by server software updates, incorrect library configurations, problems with file system permissions)
- Common coding errors (e.g. `ArrayIndexOutOfBoundsException` exceptions, unused method parameters, SQL syntax errors, illegal references from JSP pages to Java classes)
- Third-party bugs (e.g. JVM errors, issues with external data provider data, problems with external libraries)
- `NullPointerException` (NPEs)
- Concurrency bugs (e.g. deadlocks, race conditions)

FindBugs implements a lot of similar functionality to BugBot but is limited to the Java programming language and requires user download and installation. The program also admits to requiring a lot of heap space and a relatively fast CPU to function correctly on a user's local device. FindBugs is further analysed in section 5.3.2 as a method of expanding BugBot's capabilities.

In a case study carried out in 2012 by Vestola of Aalto University to evaluate and enhance FindBugs [32], FindBugs was put to the test on a project that had a known total of 132 bugs, Vestola estimated that 9% of these bugs could be caught using static code analysis. However only 1.5% of these bugs were caught.

Furthermore, in a study by Kim and Ernst in 2007 [33] who focused on the precision and recall of FindBugs, JLint, and PMD in three candidate programs found that FindBugs precision⁴ was 5–18% from these three when including all warning priorities.

These studies highlight the limitations of FindBugs as a static analysis tool and comment on FindBugs' inclusion of a number of false positive inducing bugs. Wagner et al. in a 2008 [34] study evaluated the cost of configuring FindBugs to analyse for fewer than half of the available bugs – the critical bugs, to be 7 hours-worth of work and 420euro.

2.5.4 Commercial Static Analysis Companies

There are many companies that offer commercial static code analysis; Coverity [35] and RogueWave Softwares' KlocWork [36] among many others. These companies target enterprise level coding projects. The main issue with commercial code analysis companies is cost, Klocwork costs \$20,000 annually for projects of up to a half-million lines of code [37] they can also come with license agreements forbidding the publication of any experimental or evaluative data [38].

2.5.5 Coverity Scan

Coverity Scan is a service of Coverity that operates on remote open source coding projects that are saved to online git repositories such as GitHub. It scans for a collection of over one thousand bugs. Coverity reports that more than 3700 open source projects and 16000 developers use their static code

⁴ Precision is the number of actionable warnings produced by the static code analysis tool.

analysis service [39] and as FindBugs operates under a LGPL Coverity now incorporates FindBugs in their scanning software.

Coverity Scan describe the manual process that a user must carry out to have their code scanned as follows:

- Add Coverity Scan plugin to your build process
- Register your project with Coverity Scan to get the Project token
- Sign-up or Sign-in to Coverity Scan
- Register your project with Coverity Scan.
- After registering your project, copy the "Project token" found under "Project settings" tab
- Enter the "Project token" and notification email in Coverity Scan plugin
- Additionally, this the process included the following steps that were left unmentioned:
 - Download the Coverity Scan Self-Build tool
 - Build the project and upload a .tar file to Coverity Scan
 - Define and Configure components for Coverity Scan to group defects
 - Create and upload a modelling file, this is a file specifying the behaviour of those interfaces that are linked in from code, but is not compiled and analysed.

This, in addition to agreeing to a lengthy terms and conditions agreement, makes Coverity Scan a time-consuming process that requires a number of Coverity-Scan specific additional steps in the development process.

2.5.6 Lint

Lint was the original static code analyser for C but is part of the lint family of code analysis tools for a variety of programming languages such as Java- JLint, C++- cppLint, Python- pyLint and JavaScript- jsLint.

JLint [40] is a similar Java code analyser service to FindBugs. It analyses Java bytecode and performs both syntactic pattern matching and data flow analysis on its candidate code. It searches for inconsistencies and synchronization problems by carrying out data flow analysis on the code and building the lock graph JLint claims to detect three major issues, synchronisation – deadlock and race conditions, inheritance - components shadowing superclass variable names, and data flow issues

JLint is not currently actively developed. The latest version 3.1.2 is from January 2011 and according to the tool's change log, no major changes have been made to the tool after the release of the version 3.0 in June 2004.

2.5.7 LAVA

LAVA which stands for Large-scale Automated Vulnerability Addition; is a study based on measuring false negatives of debugging programs [41]. LAVA researchers believe there is already a way to measure false positives of debugging programs; by analysing the bugs that the debugging software has pointed out and declaring the results true or false. But there exists no way of measuring false negatives i.e. if a debugging software points out 15 bugs in a program and after developer-analysis it is believed all are truly bugs, there remains no confidence that this was 100% of the bugs in the program or 1% of the bugs in the program. LAVA's way of determining the false negatives is to inject code with a specified number of bugs and measure the performance of current bug-finding tools. This strategy is applied to this project as a means of testing the performance of the Bot.

2.5.8 Summary of Static Analysis Tools

The table below highlights the different features of each mentioned static analysis tool.^{5 6}

Comparison of Static Analysis Tools							
	Multiple Languages	Free	Open Source	Actively Seeks Bugs	Remote Repo Capabilities	Download & Installation	Requires Resources
BugBot	✓	✓	✓	✓	✓		
FindBugs		✓	✓			✓	✓
Jlint		✓	✓			✓	✓
Converity Scan	✓	✓			✓	✓	✓
KlocWork						✓	✓

Table 1 Comparison of Static Analysis Tools

From table 1 it is evident that BugBot is a unique service unmatched by the existing static code analysers in terms of its usability and capabilities. Although a range of free and open source static code analysers exist all require download and installation of some form and none actively seek bugs among remote repositories. Coverity Scan is the only other static code analyser that services remote repositories but as detailed in section 2.5.5 the process of requesting a scan is time consuming and requires download of a Coverity build tool.

⁵ Note: Although Lint and KlocWork can be used for multiple languages, each language requires its own version of Lint tool or KlocWork Subscription.

⁶ Required Resources refers to either financial resources or CPU/memory usage on a user's local device.

2.6 TECHNOLOGIES USED

The development of BugBot requires a range of technologies. The main technologies I used to develop certain aspects of BugBot are listed below.

2.6.1 Java Project Using Maven



For the back-end code functionality I used the NetBeans IDE [30] and created a Java project using Maven [42]. This is due to familiarity and Mavens ease of adding open-source projects/ dependencies. Java offers a range of useful libraries that will be utilized for this project such as the JGit, File and Regex frameworks.

Maven is a software build tool which can manage the project build, reporting and documentation. Maven standardises all builds and manages any Java based project, often compared with Apache ANT [43].

2.6.2 Database Technology



To store all bugs and their attributes as well as a Queue and Record table for repository URLs a MySQL[10] database was compiled and linked to the back-end code. Java uses a JDBC to connect to databases. This comprises both the JDBC API and the JDBC Driver Manager which the program uses to connect to the database. JDBC allows you to connect to a wide-range of databases (Oracle, MySQL etc.), or to simply use the in-built database with the Java/NetBeans software. Because the database for this project is linked to both the back-end code and administrator's GUI the best option was MySQL which allowed easy manipulation of the data through a Visual Studio's Form and easy connectivity of data from back-end code created on NetBeans in the Java programming language.

2.6.3 Online Code Repositories



BugBot depends on online code repositories for operation.

Due to popularity and usability the bot targets git repositories. This includes sites; GitHub and Bitbucket and can be extended in the future to include sites such as SourceForge [44] and CloudForge [45].

GitHub ban the use of web-crawling bots unless you receive authorisation from GitHub approving of your web-crawling. That being said, direct requests from GitHub users are completely valid as this is authorised.

Bitbucket is another widely used online code repository that fortunately does not ban the use of bots. Bitbucket is the anchor URL for the autonomous web-crawling of BugBot.

The robots.txt standard is a method for remote repository users to opt-out to bot use by including a robots.txt file in their repository along with a list of directories that they disallow robot entry to. It defines a policy for search bots that signals whether or not a certain directory can or cannot be indexed by a robot [46].

2.6.4 Web Crawler

BugBot implements web crawling to navigate through git repositories. This is implemented as a separate java class that requires an anchor URL and crawls from there. Web-crawling using the Java programming language can be implemented using the JSoup Class and an iterative method to branch out to leaf directories containing the git repositories.

Another notable point of web-crawling is that successful web-crawling requires an additional database to store the previously visited pages so as not to keep entering the same web-page. This utilises the MySQL database.

2.6.5 Visual Studio and Visual Basic



Visual studio [47] with Visual Basic was used for the BugBot administrator's GUI; this was due to my own familiarity with VB and the ease of creating GUIs on Visual Studio. This GUI acts a local endpoint of the bot and will allow for easy monitoring of bot performance and additions to the bug database. Visual Studio's ability to communicate with Microsoft Azures [48] bot emulator will make adding to/editing the database and displaying bot statistics and progress tracker relatively simple if Azure is implemented to run BugBot in the future. Accessing a live feed of the twitter profile can also be done using visual studio.



2.6.6 Website Creation

The website is for direct request BugBot scans, offering user information and suggesting or reporting a bug to the bug database. Website creation requires the purchase of a domain name and a hosting service. The domain name and hosting was purchased from GoDaddy [49]. To create a user-friendly user interface of the website I used WordPress [50] as it's free, easy to use and hugely extendable. WordPress has a vast range of Plugins, one of which; ContactForms, was utilized for the direct request feature.

CHAPTER 3 DESIGN & IMPLEMENTATION

3.1 REQUIREMENTS

3.1.1 Design Requirements

The Design Requirements outline the requirements from a relatively top-level and basic perspective. They identify the overall system goals to the user and administrator from a functional and quality of service standpoint.

DR1. User-friendly and Transparent Service

Due to the malicious use of bots on code repositories in the past, it is important that BugBot implements an attentive and friendly approach to helping developers fix their programs and not irritate them with useless and impractical “fixes”. The users should be made completely aware of the bug found and the fix implemented in order to maintain a level of transparency and trust.

DR2. Minimal Time and Effort Required by User

One of the main features of BugBot that differentiates it from similar static analysis tools such as FindBugs, is the time and effort, or rather, lack-of time and effort involved in using the service. It does not require the download and installation of a plug-in or to pause development while debugging the code. BugBot is designed with this in mind.

DR3. An Easily Manageable Collection of Bugs

As there are numerous variations of candidate bugs that, after extensive testing, could be added to the BugHive, the collection must be easily manageable i.e. edited/ removed/ appended.

DR4. An Expandable System

This requirement is closely linked with DR3. The way in which BugBot is expanded is by addition of bug definitions, this requirement is supported by the easily manageable collection of bugs in addition to a dynamic basis capable of implementing more with few adjustments.

DR5. A Method of Viewing Performance and Progress

With a project such as this, it is important to be able to monitor BugBot's current performance and its performance over time. The Bot should have statistics and reports readily available to the administrator and third-party users. This encourages traceability of issues and the growth and development of the project.

DR6. Provide Information to Users of Bugs Found

BugBot can be used as an educational tool. For beginner programmers it may be unclear why a bug was labelled a bug and in turn fixed even with a short commit message to outline it. Every bug can be explained in greater detail than a short commit message. BugBot should make this information readily available to users to inform them fully of the bug encountered.

3.1.2 Technical Requirements

The technical requirements concern the more technical low-level aspects of the project such as the back-end code, database implementation and navigation/ web-crawling concerns.

TR1. Dynamic and Comprehensible Source Code

For BugBot to be transparent and expandable as mentioned in design requirements DR1 and DR4, the source code must also be dynamic and comprehensible. BugBot should be dynamic enough to handle the implementation of a bug immediately after its addition to an external database with minimal change to the source code. As BugBot is to be an open source project a level of comprehensibility is also required in its structure and implementation.

TR2. Identify & Fix Bugs Correctly Using Regular Expression

The key aspect of BugBot is its correct identification of bugs in a variation of programming languages. This is to be achieved through Java's regular expression class. For the successful identification and fix of a bug, the bugs must be defined with all necessary criteria i.e. Regular Expression syntax, appropriate Fix, Commit Message etc.

TR3. Extensive Testing to Avoid False Positives.

Each bug must be extensively tested in its regular expression form to avoid false positives. This requires a test class with a sole purpose of matching regular expression to bugs in context.

TR4. A Platform for Users' Direct Requests

Direct requests are requests made by users to visit their code repositories whereby the URL of the repository would be indicated/supplied to the Bot. The bot should then handle the request seamlessly and in good time without disrupting any process it is currently working on.

TR5. User Ability to Report & Add Bugs

Bugs must be provided to users in a clear and constant format. Users should also be able to add bugs to the BugHive easily to encourage the constant growth and evolution of the bot. Reporting functionality is also a key aspect of BugBot to ensure accountability and traceability of issues.

TR6. Navigate Queue and Record Databases

For continuous and seamless operation Record and Queue databases are required to organise the Repository URLs that are to be scanned and the Repository URLs already scanned. These databases then need to be accessed and added to periodically during operation.

TR7. BugHive Database

A BugHive database is required to distinctly define all criteria of a bug and for the easy accessibility of the back-end code to the necessary information required at any given time The BugHive database will contain all “live” bug definitions that are to be deployed on the scanned repositories

3.2 SYSTEM COMPONENTS

BugBot as a system comprises many parallel processes to operate as intended without fault in order to fulfil all aforementioned design and technical requirements. Namely:

- The Back-end code
- Configuration & Data files
- MySQL Databases: BugHive, Queue, Record
- Website: www.BugBot.org
- Administration GUI
- Twitter Account: @BugBot16

These components need to work seamlessly with each other for BugBot to be manageable, accountable and successful in fulfilling its objectives.

Figure 9 is a basic top level visual of how each of the components, mentioned above interact. This is described in detail throughout this section.

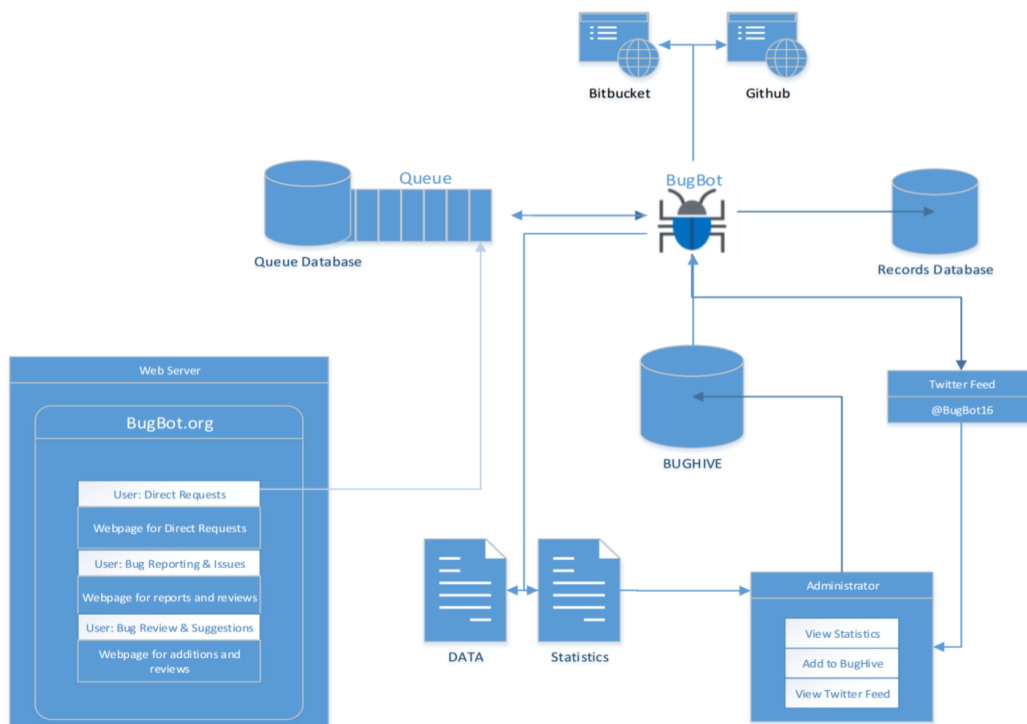


Figure 9 Top Level System Interaction

3.3 BACKEND CODE

The back-end code of BugBot was written in Java. Below is the basic operation of the back-end code without distinguishing classes and methods.

3.3.1 Top Level Code Flow

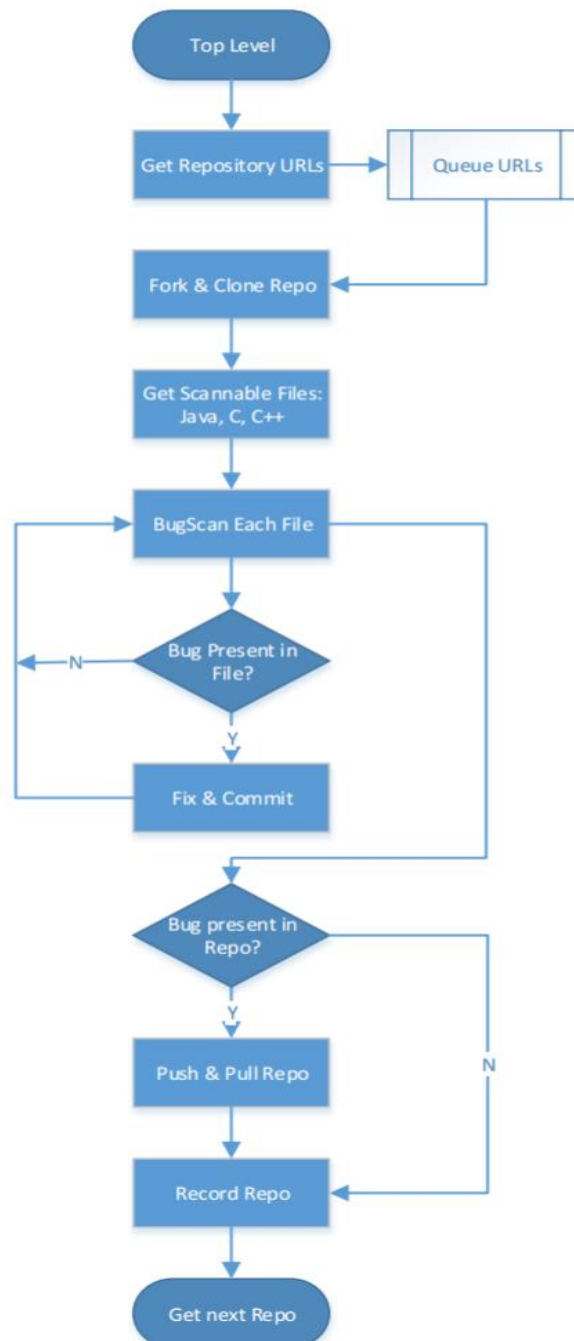


Figure 10 Top level Code Flow Chart

All code is open source and available at: TBC

3.3.2 Classes & Corresponding Key Functions

BugBot's backend code comprises a total of 11 Java classes. Each class has a specific function to correspond with each of the many components of BugBot's functionalities.

3.3.2.1 Driver Class

The Driver class contains one method; the main. This links to all other classes and methods. The key functionality operates within an infinite while loop to continuously conduct scans of the repository at the top of the queue.

If the queue length drops below a pre-defined threshold the while loop diverts to refilling the queue.

This is done by calling the following methods in priority order:

- Priority 1: Get Website Direct Requests
- Priority 2: Get GitHub BugBotFINDME.txt requests
- Priority 3: Web-Crawl BitBucket

When the queue is replenished, the code reverts to creating a RepoScan object with the URL at the top of the queue.

```
int run =0;
while(run < numScans)
{
    /***** KEEP QUEUE LENGTH > minQlen *****/
    Qdb = new QueueDB(); //initialise new connection to queue to avoid time-out exception
    int Qlen= Qdb.getLength(); // get length
    admin.setQlength(Qlen); // set the length in the BugBotConfig file

    /***** IF QUEUE LENGTH IS less than THRESH *****/
    while(Qlen<minQlen){
        try {
            System.out.println("length of Queue: " +Qlen);
            admin.setStatus("Finding");

            /***** GET DIRECT, INDIRECT, BUGBOT.ORG REQUESTS *****/
            WebsiteDirectRequest.getEmails(); //priority 1
            WebCrawlgithub directRequests = new WebCrawlgithub(); // priority 2
            WebCrawler indirectrequests = new WebCrawler(); // priority 3

            Qlen= Qdb.getLength();

        } catch (IOException ex) {
            Logger.getLogger(Driver.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /***** GET URL AT TOP OF QUEUE *****/
    url = Qdb.getFirst();
    admin.setCurrentRepo(url);

    /***** REPOSITORY SCAN URL *****/
}
```

Figure 11 Driver Code Before Creating a RepoScan object

From here, the RepoScan class conducts all the Git and bug scanning functionality until the scan of the repository is done. This is when the Driver class will remove the URL from the top of the queue and “record” it in the Records database. And thus, return to the top of the while loop to conduct the next scan. “numScans” and “minQlen” are both private static final integers, set before running the code they are usually set to 100 and 10 respectively.

3.3.2.2 *RepoScan class*

This class carries out all fork, clone, and commit git functions as well as the bug scanning methods.

The method begins by forking the repository and cloning to the local device. Once cloned it retrieves and saves the path of all scannable files i.e. the JAVA, C and C++ files, in a list. It then begins scanning each line of code of each file.

Scanning the file consists of:

- Identifying and storing declared variables.
- Identifying and storing undeclared variables
- Identifying bugs with the pre-defined regular expression.

This functionality is further discussed in sections 3.3.5 and 3.3.6 respectively.

If a bug is found a “bugPresent” flag is set and the RepoScan class will subsequently fix the line, fix the file, and commit the fix. Once the entire repository has been scanned the commits are then pushed and a pull request is made.

The RepoScan class is uniquely programmed to deal with just the Java programming language files, all other files are effectively “sent” to another RepoScan class “RepoScanC” to deal with the C bugs; both C++ and C. The RepoScanC class carries out most of the same scanning functionality as the RepoScan class i.e. scanning each line of each C file, setting the variables and fixing the line.

3.3.2.3 *WebsiteDirectRequests class*

This class deals with the direct requests received from the BugBot.org/requests form that users paste their repository URL into and submit on the website. This form is configured to send an email with the submitted content; the URL, to a BugBot email address. This class retrieves the unread emails containing URLs and ensures they are of the correct format. Once it has retrieved and confirmed a URL it adds it to the queue database with the time and date stamp.

This class is called from the Driver class and has priority over all other queue filling operations.

3.3.2.4 *WebCrawlGitHub class*

This class is also concerned with direct requests from users. It scans GitHub for the BugBotFINDME.txt files and adds the repositories that have this file inclusion to the queue. Again, before adding the URL to the queue it ensures the correct format.

This class also extracts the metadata from the HTML of the repository page. It extracts the last time and date the file was committed and checks the Records database for when, or if, the repository has been scanned before. By comparing these times, it can ensure continuous scanning of a repository containing the file if there has been a new commit since it was last scanned. It does so by adding the commit time as the timestamp associated to the record of the repository.

This class is called from the Driver class. It is of priority 2 i.e. called after the website direct requests but before the Autonomous crawling of BitBucket repositories. The navigation involved in web crawling GitHub is further discussed in section 3.5.2.

3.3.2.5 *WebCrawler class*

This class deals exclusively with BitBucket repositories. As BitBucket make it difficult to attain a list of all public repositories they host; this class is programmed to Google search the following:

Home site: `bitbucket.org`

This returns a Google search list of approximately 70,000 repositories. Each call to this class reads exactly one page of Google search results and so methods exists to both save and read the current “bookmark” of the page previously scanned. This bookmark is saved in the BugBotconfig.txt file and accessed at each entry of this class.

This class is priority 3 of filling the queue and so is called from the driver class when the queue depletes below a certain threshold and direct request checking classes have been called. The web crawling involved in collecting the URLs from Google search is further discussed in section 3.3.8.

3.3.2.6 *QueueDB class*

This class creates a connection to the MySQL ‘Queue’ database and contains all getter and setter methods to implement SQL commands on the database from the Java program.

These methods include; getLength, insertINTO, getFirst, and removeFirst.

Each database table was represented as a Java constructor class whereby the constructor created a connection to the table and all accessor methods executed various SQL statements. For example, retrieving a URL from the FIFO Queue.

```
//returns the URL of the first repository in the queue
public String getFirst()
{
    String url=null;          //URL as String
    ResultSet rs;
    try {
        rs = runSql("SELECT * FROM Queue");    //selects entire Queue
        if(rs.first()){                        // get first result
            url= rs.getString("URL");          // set url
        }
    } catch (SQLException ex) {
        Logger.getLogger(QueueDB.class.getName()).log(Level.SEVERE, null, ex);
    }
    return url;
}
```

Figure 12 Accessor method getFirst() of Queue Database

3.3.2.7 *CrawlDB class*

This class creates a connection to the MySQL 'Record' database. And contains all getter and setter methods to implement SQL commands on the database from the Java program. The main method of this class is to insert a scanned URL into the Record database with a timestamp.

3.3.2.8 *BugHive class*

This class creates a connection to the BugHive database. As the BugHive database contains bugs defined with fourteen different criteria the BugHive class contains getter methods for each of these criteria given the bug number as a parameter. This allows for easy scanning of each line of a file in a for loop where the loop executes up to the database length.

```
public String getRegEx(int i)
{
    String reg=null;    // the regular expression pattern returned as Sttring
    try {
        rs = stmt.executeQuery( "SELECT * FROM BugHive WHERE Num=" +i+ ";" );
        // returns a result set of 1 as numbers are unique identifiers
        if(rs.next()){
            // returns the Regular Expression value of the element in RS
            reg = rs.getString("RegularExpression");
            //System.out.println(reg);
        }
    } catch (SQLException ex) {
        Logger.getLogger(BugHive.class.getName()).log(Level.SEVERE, null, ex);
    }
    return reg;
}
```

Figure 13 *getRegEx() Method of the BugHive class*

3.3.2.9 *Admin class*

For all administration purposes, BugBot requires a BugBotConfig.txt file. This file contains the details as outlined below. The figures shown are the default values of which BugBot was reset to during debugging.

- GooglebitbucketSearchPage=1
- FilenameCount=1
- ReposAccessed=0
- RepoAccessRequests=0
- bugsFound=0
- changesPushed=0
- bugsNum=0
- CurrentRepo=Nothing
- Status=Finding
- QueueLength=0
- origBugsNum=12
- j_cmr=0
- j_ecs=0
- j_aiv=0
- j_irv=0
- j_ioi=0
- j_idN=0
- j_abc=0
- j_wabc=0
- c_avc=0
- c_did=0
- c_irv=0
- c_wavc=0

This is for statistical and monitoring purposes which are further described in section 4. At each important and relevant step of BugBot's process the configuration file is updated correspondingly, this is done through the Admin class via numerous setter methods.

3.3.2.10 *Tweet class*

This class is for the control of BugBot's Twitter account. Depending on new statistical updates this class creates and posts a relevant tweet.

```

public void addReposAccessRequests(String url) {
    Tweet t = new Tweet("BugBot has been requested to scan: " + url +
        ", for BugBot to scan your' repo go to www.BugBot.org/BugBot");
    this.reposAccessRequests++;
    updateAdminFile();
}

```

Figure 14 Tweet Created from the Admin class

For the successful implementation of this functionality the Twitter4j framework was used. To authenticate requests made from BugBot to create and posts tweets an application consumer key and secret, and an access token key and secret were obtained. This is part of Twitters' application permission model.

Tweets are made at milestones such as:

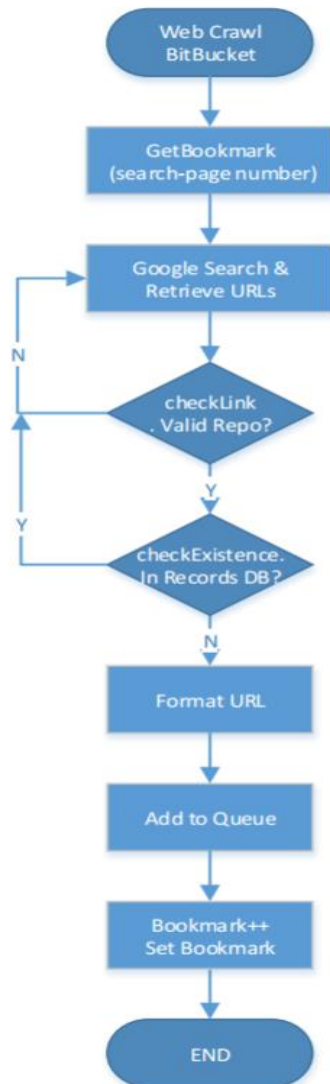
- A direct request being made.
- A bug added to BugHive.
- BugBot Accessing 100 Repositories.



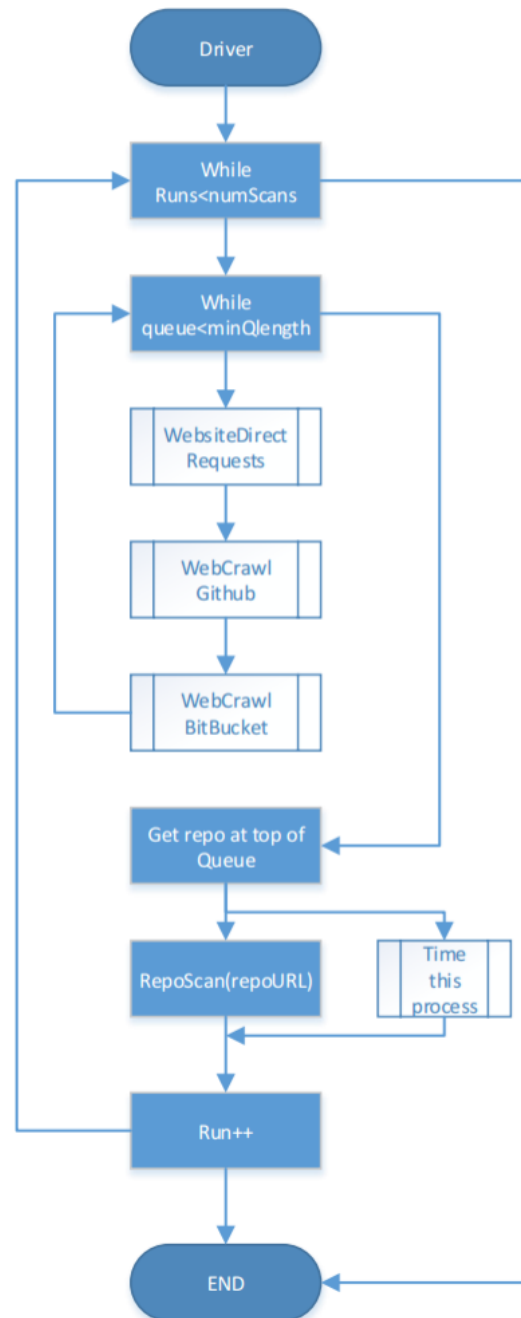
Figure 15 BugBot Twitter Feed

3.3.3 Code Flow Charts by Class

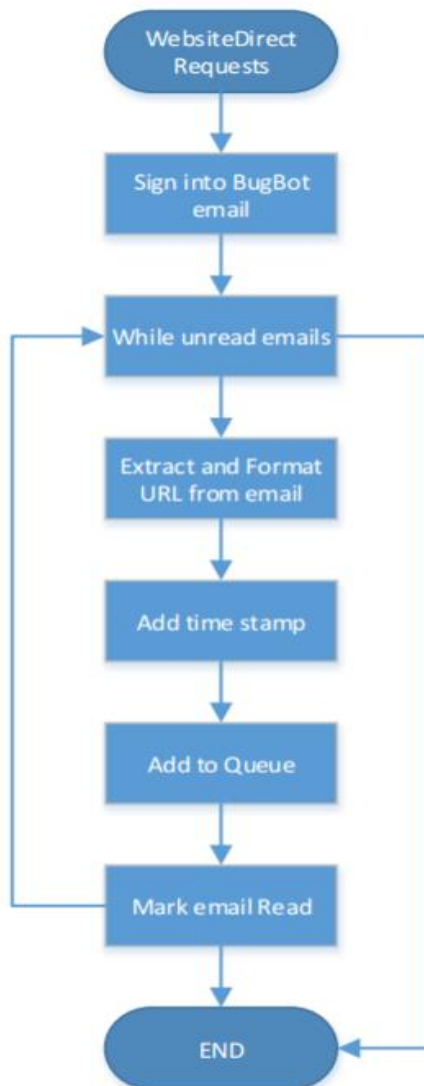
3.3.3.1 Driver



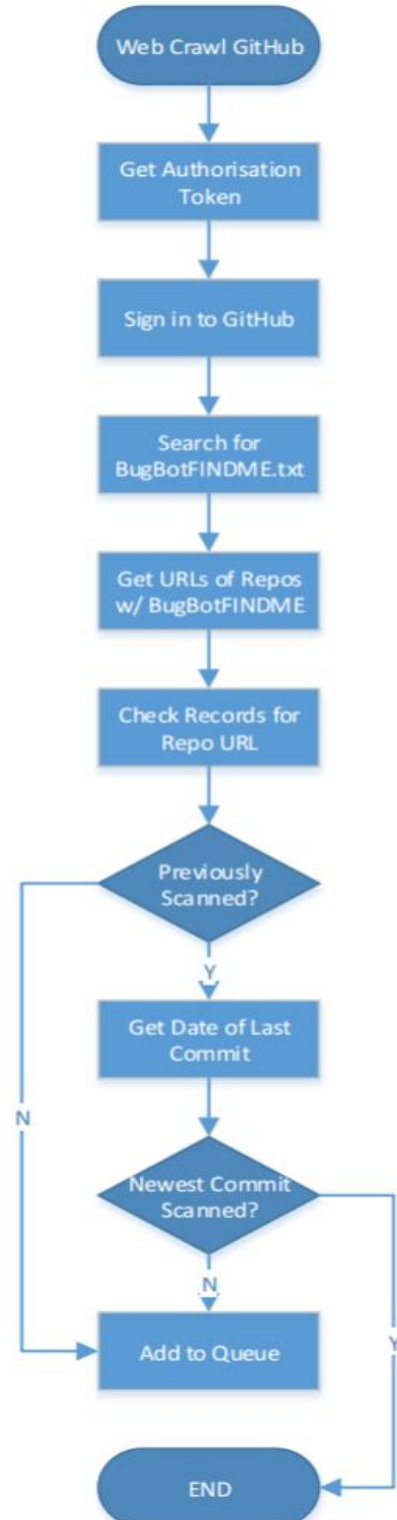
3.3.3.2 WebCrawler BitBucket



3.3.3.3 Website Direct Requests



3.3.3.4 WebCrawl Github



3.3.4 Backend Code Analysis

With the use of the software metrics tool Source Monitor [51], BugBot's backend code was analysed at baseline and again after reviewing and editing the code. Source Monitor calculates code metrics and displays useful values such as the maximum and average cyclomatic complexity of a method, number of methods per class, and number of statements per method. These metrics allow insight into which methods and classes are most susceptible to bugs or errors but do not point out where the bugs and errors might occur.

Cyclomatic complexity is a quantitative measure of the number of linearly independent paths through a program or method's source code [52]. It has been hypothesised that limiting cyclomatic complexity of methods to 10 or less is a successful approach to avoiding errors, incomprehensibility and hard to modify code. This limit has significant supporting evidence, but limits as high as 15 have been used successfully as well [53].

3.3.4.1 Baseline

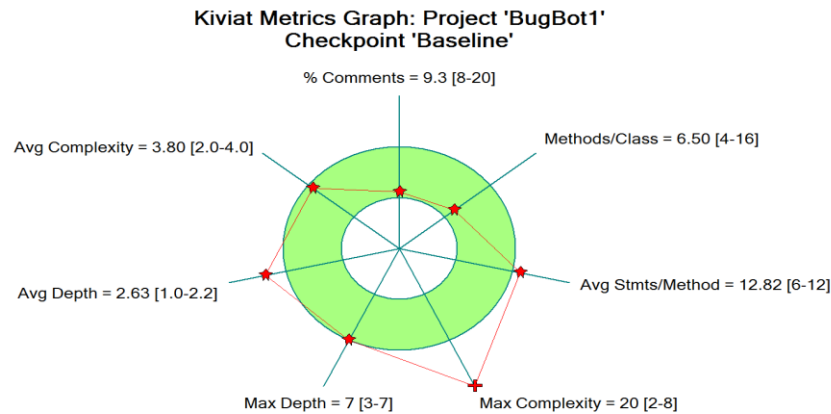


Figure 16 Kiviat Metrics Graph at Baseline

At baseline it was observed immediately that the complexity of some methods was much too high with a maximum complexity of 20, as a general rule a complexity greater than 10 is a danger zone especially if no unit tests exist for the method. It was also observed that the percentage of comment coverage was less than 10% which is relatively low for a project which aims to be comprehensible.

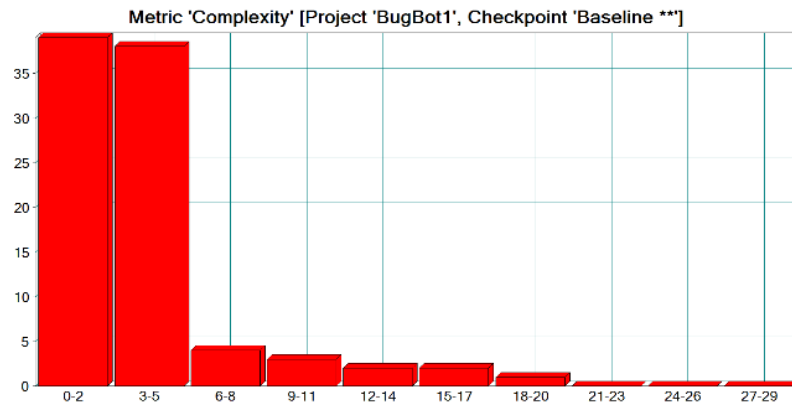


Figure 17 Baseline Metric Graph

From analysing the metrics summary report as in figure 17, it was decided the program would be reviewed and edited to provide less complex and more comprehensible source code. The aim was to increase the comments to at least 15% and reduce complexity to a maximum of 10.

Metrics Summary For Checkpoint 'Baseline' (Printed on 13 Mar 2017)

Parameter	Value
Project Directory	C:\Users\user\Documents\NetBeansProjects\fyppgit\
Project Name	BugBot1
Checkpoint Name	Baseline
Created On	13 Mar 2017, 16:39:35
Files	11
Lines	2,416*
Statements	1,526
Percent Branch Statements	15.8
Method Call Statements	1,011
Percent Lines with Comments	9.3
Classes and Interfaces	14
Methods per Class	6.50
Average Statements per Method	12.82
<hr/>	
Most Complex Methods in 14 Class(es):	Complexity, Statements, Max Depth, Calls
?(instance of FileReader).br.readLine()	1*, 1, 4, 2
...\?(instance of javax.mail.Authenticator).getPasswordAuth...	1*, 1, 5, 1)
Admin.checkAdminFile()	20*, 61, 6, 65
BugHive.getVariations()	5*, 13, 4, 6
CrawlDB.executeUp()	4*, 10, 4, 8
Driver.main()	9*, 46, 7, 35
QueueDB.finalize()	3*, 2, 3, 2
RepoScan.recordVars()	16*, 80, 6, 54
RepoScanC.extractVarsRegex()	5*, 7, 4, 13
Tweet().twitter.setOAuthAccessToken()	1*, 1, 3, 1
Tweet.Tweet()	2*, 2, 3, 1
WebCrawler.getBookmark()	5*, 15, 6, 15
WebCrawlgithub.getCreateDate()	5*, 17, 6, 12
WebsiteDirectRequest.getEmails()	1*, 5, 2, 1

3.3.4.2 Post Review and Edit

As the most complex methods per class were provided in the metrics summary above, the first method to be edited was the Admin.checkAdminFile method. This method had the responsibility of extracting all 8 values from the BugBotConfig file and so to resolve the issue each of these values were retrieved in separate getter methods. This immediately reduced the complexity of the method to 2 and all consequent getter methods were of complexity 4.

Multiple methods in the RepoScan class were also broken up to handle less functionality. The results of 15 total reviews and edits or “Checkpoints” are shown below.

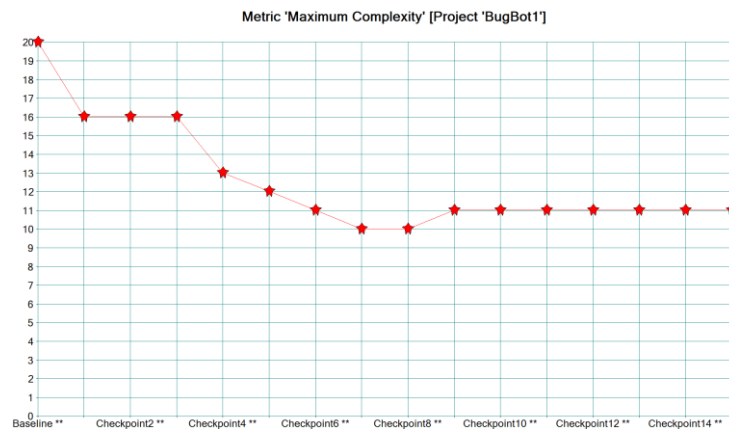


Figure 18 Reducing complexity over 15 Checkpoints

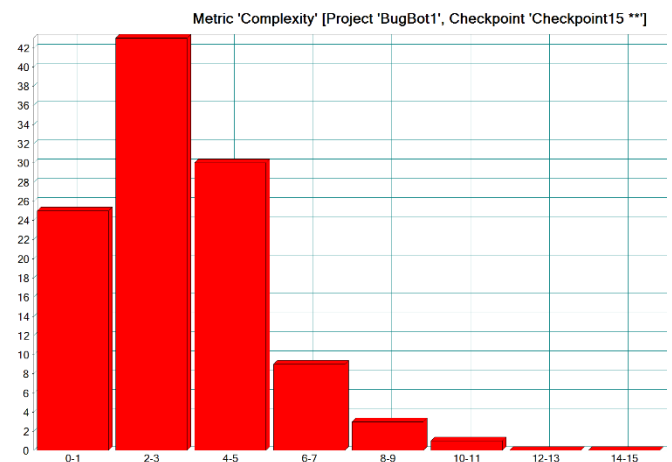
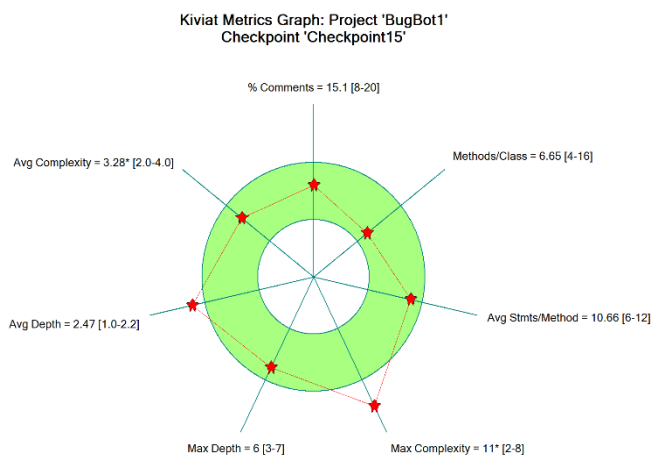


Figure 19 Post Review & Edit Kivi and Complexity Distribution Graphs

Metrics Summary For Checkpoint 'Checkpoint15' (Printed on 20 Mar 2017)

Parameter	Value
Project Directory	C:\Users\user\Documents\NetBeansProjects\fyppgit\
Project Name	BugBot1
Checkpoint Name	Checkpoint15
Created On	20 Mar 2017, 18:01:54
Files	13
Lines	2,629*
Statements	1,585
Percent Branch Statements	15.5
Method Call Statements	1,050
Percent Lines with Comments	15.1
Classes and Interfaces	17
Methods per Class	6.65
Average Statements per Method	10.66
<hr/>	
Most Complex Methods in 17 Class(es):	Complexity, Statements, Max Depth, Calls
?(instance of FileReader).br.readLine()	1*, 1, 4, 2
\...\?(instance of javax.mail.Authenticator).getPasswordAuth	1*, 1, 5, 1)
Admin.updateAdminFile()	11*, 45, 5, 37
Book.getTitle()	1*, 1, 2, 0
BugHive.getVariations()	5*, 13, 4, 6
ComparatorImpl.compare()	3*, 8, 3, 9
CrawlDB.executeUp()	4*, 10, 4, 8
Driver.main()	7*, 40, 5, 31
Library.describeBooksBy()	3*, 5, 3, 7
QueueDB.finalize()	3*, 2, 3, 2
RepoScan.assignedVar()	9*, 36, 6, 26
RepoScanC.extractVarsRegex()	5*, 7, 4, 13
Tweet().twitter.setOAuthAccessToken()	1*, 1, 3, 1
Tweet.Tweet()	2*, 2, 3, 1
WebCrawler.getBookmark()	5*, 15, 6, 15
WebCrawlerGithub.getCreateDate()	5*, 17, 6, 12
WebsiteDirectRequest.getEmails()	1*, 5, 2, 1

3.3.4.3 Discussion

Although the maximum complexity is still over the desired limit of 10, the method with this complexity value; Admin.updateAdminFile, was thoroughly reviewed and no further changes could be made. The method was however reduced from complexity 16 to 11.

The percentage of comments was increased from 9% to 15.5% in an effort to increase comprehensibility. This was the result of explaining complex methods and adding method summaries as headers of various key methods.

BugBot's backend code comprises a total of 11 classes, 2,629 lines of code and 113 methods.

3.3.5 Variable Identifying & Analysis

For the successful identification of variable dependent bugs, BugBot is responsible for identifying and analysing variable types; String, int, long, double, char and Boolean. This is done before each line is bug-scanned. As most bugs in the BugHive are variable dependent i.e. their identification depends on what kind of variable is involved, the variables in a file are identified and stored for future parsing.

There are two types of variables to identify; declared and undeclared as below in figure 20.

```
String s= "hi";
String s1 ="hello";
if(s.equals(s1))
{
    // both s and s1 are declared variables
}
if(s.equals("hey"))
{
    // 'hey' is not declared a String, BugBot must identify it as a String
}
```

Figure 20 Declared and undeclared variables

Every variable, declared or undeclared is stored in a two-dimensional variable array of width 3. This takes the form of:

Variable Type	Variable Name	Variable Value	Class
int	i	0	declared
double	d	0.4	declared
String	s1	"hello"	declared
String	s	"hi"	declared
String	[string]	"hey"	undeclared

Table 2 Visual of Variable Array Structure

The length of this array is dependent on the number of variables in the file.

These variables are recorded in two separate methods; recordVars and recordDefaultVars within the RepoScan class and are identified using Java's regular expression class.

The recordDefaultVars method is to identify and store undeclared variables. There are three different resulting methods used to do this as shown below in figure 21.

```

// undeclared variables
private void recordDefaultVars(String line)
{
    //get integers or long variables
    getUndeclaredintlong(line);
    // get double or float variables
    getUndeclareddoublefloat(line);
    //get character or String variables
    getUndeclaredcharString(line);
}

```

Figure 21 Method to Record Undeclared Variables

Each of these methods has its own regular expression pattern to identify either of the two variable types, and a check of which of the two types the matched pattern is.

Regular Expression	Variable Type	Extra check
\d+	int or long	✓
((\d+)(\.)?(\d+))	double or float	✓
\".*?\"	String	
\'.?\'	char	

Table 3 Regular Expression to Identify Variable Types

To identify and store declared variables the following method was called and three resulting methods were called from this for: multiple declarations of variables on one line, the declaration and assignment of one variable on a line, and the declaration of one singular variable on a line without assignment.

```

// declared variables
private void recordVars(String curline)
{
    multDecVariables(curline); // e.g int i,j,k;
    assignedVar(curline);      // e.g int i=9;
    noAssignVar(curline);      // e.g int i;
}

```

Figure 22 Method to Record Declared Variables

The regular expression patterns for these types of variables are much more complex and required extensive testing. For further analysis of each repository, the number of declared (Java only) variables are recorded in a DATA.csv file

3.3.6 Bug Identifying & Fixing

Bug identification and fixing was implemented with a dynamic approach. Each bug is defined in the SQL database; BugHive with a Regular Expression form. The regular expression version of each bug is applied to each line of code in the BugScan method.

If there is a match; the line is passed to a further method called analyseBug. As the regular expression check does not inherently check the variable types in the line, analyseBug is left to take variables into account. This process is best described by example.

Line of Code: `if(string1 == string2){`

Regular Expression: `((\w+)\s*((\=\=)|(\!\=))\s*(\w)+)`

Bug Matches: `string1 == string2`

Variable Array (already populated as explained in section 3.3.5) contains two relevant entries among others:

Variable Type	Variable Name	Variable Value
String	string1	"hello"
String	string2	"hi"

Table 4 Example of Variable Array Structure

AnalyseBug parses the variable Array “Name” column to find the variables contained in this line of code. It then populates two ArrayLists as below.

ArrayList BugVars: `[String, String]`

ArrayList BugVarNames: `[string1, string2]`

The method `extractVars` is now called and passes the “General Form” of the bug as defined by the `BugHive` as a parameter. This populates the `ArrayList` `relevantVars`.

```
String form= bugHive.getGeneralForm(bugNum);  
relevantVars =extractVars(form);
```

General Form: `[string] == [string]`

ArrayList relevantVars: `[string, string]`

`BugBot` now compares the variables stored in `BugVars` and `relevantVars` through the method `compareVars`. In this example this results in a bug match and so `BugBot` proceeds to fix the line

To fix the line the “fix” as defined in the `BugHive` is retrieved.

```
String fix =bugHive.getFix(i);
```

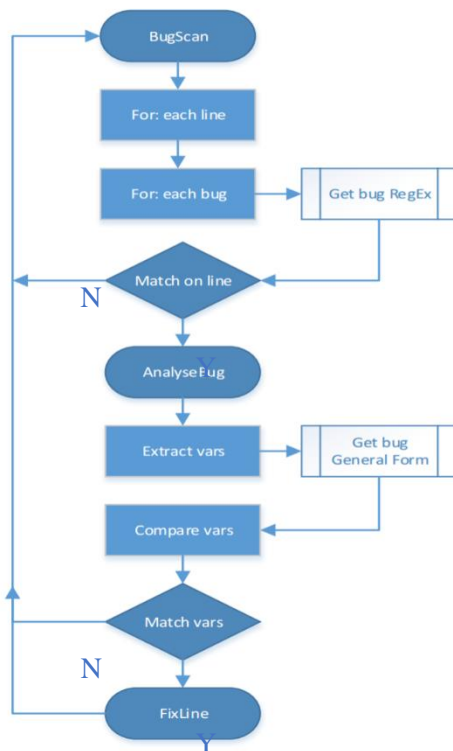
Fix: `[string].equals([string])`

Using the `setVars` method, the variable names now stored in `BugVarNames` are placed in their respective positions i.e. in replacement of the square brackets and their content.

Fixed Bug: `string1.equals(string2)`

The bug is then replaced with the fixed bug in it’s original line of code and the fixed line replaces the original buggy line.

Fixed Line: `if(string1.equals(string2)){`



After the complete identification of a bug, a bugPresent flag is set which enables a Commit at the end of the file scan and a Report and Push at the end of the repository scan.

3.3.7 Databases Queue & Record

Throughout the process of scanning one repository, that repository requires connections to all three of BugBot's databases. The connections and statements to these databases rely upon the java.sql framework.

The first connection made during a repository's lifecycle is the Record database containing the list of repositories already scanned and their corresponding timestamp. Before a repository is added to the

Figure 23 BugScan Flow Chart

with the checkExistence method, if this returns false the repository is added to the Queue database. Once added to the Queue database it then waits to be passed as a parameter into the RepoScan class. Once it has been scanned the repository is removed from the queue and added to the Record database through the finish() method.

ID	URL	TimeCreated
4261	https://BugBot16@bitbucket.org/mfrigerio17/robotic...	BitBucketGoogleSearch
4262	https://BugBot16@bitbucket.org/phaentu/abctoolbox-...	BitBucketGoogleSearch
4263	https://BugBot16@bitbucket.org/devinmartin/keeotp	BitBucketGoogleSearch
4264	https://BugBot16@bitbucket.org/Unity-Technologies/...	BitBucketGoogleSearch
4265	https://BugBot16@bitbucket.org/xerial/sqlite-jdbc	BitBucketGoogleSearch

Figure 24 Queue Database as of 28th March 2017

RecordID	URL	TimeCreated
2837	https://BugBot16@bitbucket.org/libqxt/libqxt	20170328_102021
2838	https://BugBot16@bitbucket.org/atlassian/jgit-flow	20170328_111759
2839	https://BugBot16@bitbucket.org/nsegata/metaphlan	20170328_112804
2840	https://BugBot16@bitbucket.org/pypy/compatibility	20170328_112805
2841	https://BugBot16@bitbucket.org/multicoreware/x265	20170328_112806
2842	scheduledGITHUB	20170328_112810

Figure 25 Records Database as of 28th March 2017

Each of the three databases were connected to through a constructor class whereby each instantiation of the class created a new connection to the database, the multiple instantiations of the database classes were necessary to reconnect otherwise a time-out would block access. All respective database interactions were set up as accessor methods to be constantly called upon throughout a repositories lifecycle in the BugBot system.

3.3.8 Web Crawling

3.3.8.1 BitBucket Requirements

BugBot's web crawling activity was mainly for the purpose of collecting BitBucket repositories from Google search results and also to search for BugBotFINDME.txt files on GitHub. This made use of java's JSoup library.

```
String searchStr= "Home site:bitbucket.org";
String startindex = "&start=" +Integer.toString(10*bookMark);
Document searchDoc = Jsoup.connect("https://www.google.com/search?q=" + searchStr +startindex)
    .get();
```

Figure 26 For BitBucket Web Crawling

Figure 26 is the key command to searching Google for BitBucket repositories. The “bookmark” variable is the value saved in the BugBotConfig file. It is essentially the “page” of search results when multiplied by 10 and so is got, incremented and set on each call to the WebCrawler class. Each iteration of this class gathers ten BitBucket repositories provided each of the repositories are valid.

As Google blocks web crawling and uses the speed of web navigation as a sign of web crawling this method of searching page by page avoids the issue of being blocked. This method is only called each time the queue length reaches below a minimum length threshold i.e. approximately every five to forty minutes.

3.3.8.2 GitHub Requirements

To web crawl GitHub, it was necessary to gain an authenticity token to log in in order to gain access to all search capabilities, see figure 27. This is done by navigating the HTML document of GitHub's login page and providing the necessary username and password credentials.

```
String authToken = loginDoc.select("#login > form > div:nth-child(1) > input[type=\"hidden\"]:nth-child(2)")
    .first()
    .attr("value");
```

Figure 27 Attaining an Authorisation Token on GitHub

From here BugBot can search GitHub for the BugBotFINDME.txt file by entering the name of the file in the search bar.

3.4 BUGHIVE DATABASE

3.4.1 Bug Definitions

For the correct identification and fix of a bug a number of different criteria need to be defined for each bug. These are used systematically during each bug scan at different stages of the repositories lifecycle. As previously mentioned each of the bug criteria are accessed through the BugHive classes accessor methods. For each criterion, there exists a corresponding `get()` method.

3.4.1.1 BugHive Criteria

The thirteen criteria are listed as they appear in the SQL database, and described below.

- ***BUG_ID***

A short abbreviation for quick analysis of bugs. In the form of `j_cmr` as the Java bug, Casting Math Random. These ID's are generally only used by an administrator for keeping track of and counting bugs found etc.

- ***Language***

The Language the bug applies to. Currently only two definitions exist, Java and C. This is defined as the file extension of the programming files written in the language i.e. C++ would be written as “Cpp” and Python as “py”.

- ***Variable Dependent***

A boolean value set if the bug contains variables that the fix will need to consider. For example, comparing Strings as; `if(s==s1)`, the identification of this bug will depend on whether `s` and `s1` are declared String variables, fixing the bug will also involve substituting these values into the fix; `if(s.equals(s1))`.

- ***Variations***

Variations is a Boolean value set if the regular expression definition of a bug is the same as the regular expression of another. This occurs in one instance in the BugHive and the setting of the Boolean value signals further analysis of the bug to check which bug it has actually identified.

- ***General String***

The general String format of a bug is used after the regular expression has matched on a line. This definition contains the variable types required for the definitive match of a bug. The variables required are written in square brackets; [int] or [string] in place of where the variables should occur.

- ***Regular Expression***

The regular expression form of the bug is defined using Java's regular expression class. This defining required extensive research and testing to ensure correct matching and on each line. This is further discussed in section 3.3.6.

- ***Description***

The description of the bug is the message written in the report that is committed to the bug-present repositories. It is a short description outlining the details of the bug.

- ***Wanted Cases***

In the case of a bug that could actually be intended or not necessarily a bug as defined by BugBot; the Wanted Cases criteria concerns whether or not the line of code should in fact be explicitly altered and fixed. The majority of the currently defined bugs have "no" wanted cases. However, if otherwise stated; the bug requires further analysis. For example, BugBot defines the comparison of Strings in a conditional statement using "==" as a bug, but a particularly frequent use of this is `if(s==null)`. This is, of course, not a bug and so this is defined in the Wanted Cases criteria of the bug.

- ***Extended Explanation***

As the description describes the bug at a high-level, the extended explanation describes the bug's mechanisms and possible outcomes.

- ***Solution Explained***

A short explanation of the fix provided for the bug.

- ***Fix***

Written in the same form as the General String criteria, it is simply the “fix” that is substituted in place of the bug with the variable substitutions also outlined in square brackets;

```
if([string].equals([string])).
```

- ***Commit Message***

The short commit message that accompanies the fixing of the bug when pushed to the remote repository. If more than one bug is present in a file, the commit message defaults to “BugBot made fixes”.

- ***Number***

Used to iterate through the BugHive and access all other criteria of a certain bug, the number acts as a unique identifier. As the bugs are called in a for loop on each line the “counter” integer of the for loop respectively refers to a bug in the BugHive. This is then passed as a parameter to the getter methods in the BugHive class.

3.4.1.2 Bug Regex Definitions

Java's Regular Expression class is a comprehensive and technical library for pattern matching of strings. The main functions and mechanisms used are outlined below in figure 28. In addition to regular expression to identify bugs it was also used to identify variables, both declared and undeclared.

Most Used Regular Expression Symbols	
*	zero or more occurrences
+	one or more occurrences
?	at least one occurrence
\	string literal of the next character group
\w	word characters a-z and A-Z
\d	digits 0-9
\s	whitespace

Figure 28 Most Used Regular Expression Symbols for Bug Definitions

For example,

```
((\w+)\s*((\=\=)|(\!\=))\s*((\w)+)
```

This is the regular expression syntax of the java bug; comparing strings using equals to symbols rather than the equals method. The enclosing brackets group the entire regex as each segment, must occur consecutively. The \w is followed by + as a string can be declared as one letter or multiple letters. The \s is followed by * as there may be no whitespace or some white space present before the equals to signs. The = symbols are preceded by \ as it is a string literal, this is exactly how it must appear in context. These are grouped with != and an or command as comparing two strings as != or == is a bug. The remaining regular expression is as described previously.

3.4.2 Bug Languages

Currently BugBot scans for Java, C and C++ files, Java being the most extensively defined bugs. As C++ is an extension of C; all C defined bugs are also applicable to C++ files.

BugBot extracts the metadata of each BitBucket repository accessed autonomously before cloning it, as it was considered not to clone a repository that was not labelled with a BugBot supported language. However, after testing it was obvious that many repositories can contain hundreds of files of a programming language that was not listed in the metadata.

The extraction of this data does however highlight the need for BugBot to support more languages.

From the total of cloned repositories approximately 70% were declared unsupported languages.

According to figure 29, a graph of the most popular programming languages of 2016 compiled by the IEEE [54], BugBot does support three of the top four most popular programming languages.

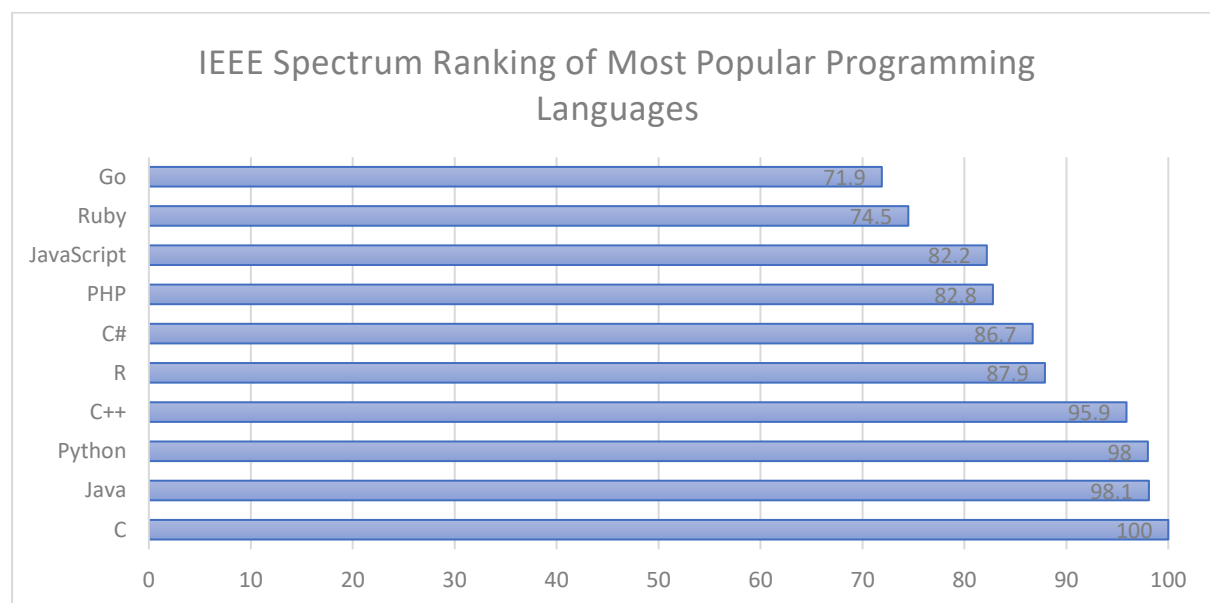


Figure 29 IEEE Graph of Programming Language Popularity [54].

GitHub's Octoverse [55] have also compiled data on the most popular programming languages hosted on GitHub by open Pull Requests. This data suggests that BugBot only supports three of the nine most popular programming languages.

From this data, it could be speculated that by adding two more programming languages; Python and JavaScript, BugBot could nearly triple the number of repositories scans.

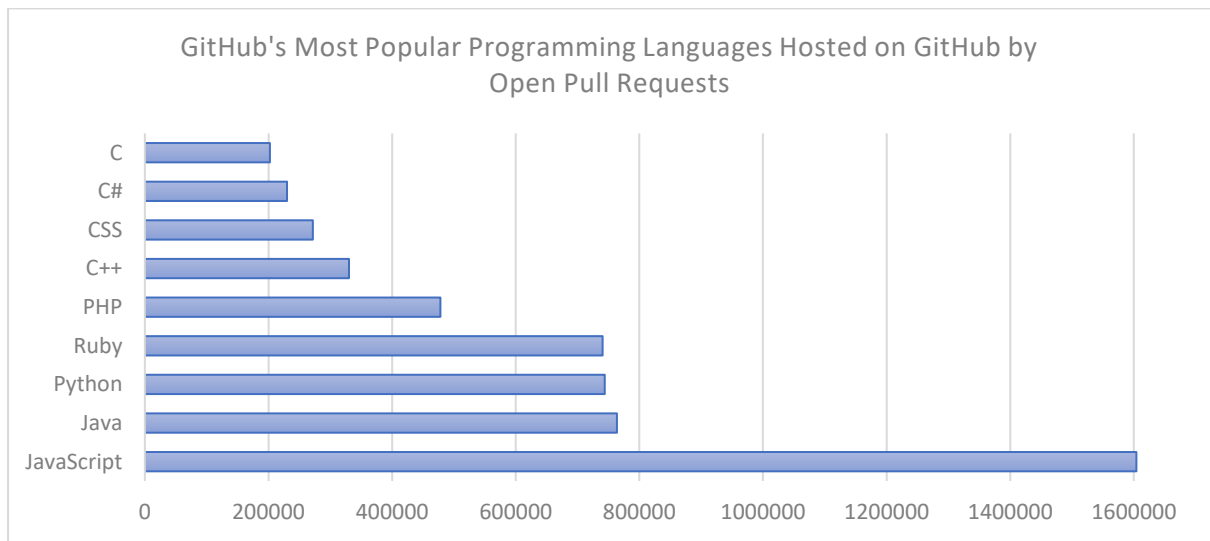


Figure 30 The Most Popular Programming Languages on GitHub [55]

3.4.3 List of Bugs

The list of bugs are available for users to view at www.ButBot.org/BugHive. The BugHive as of the 14th March 2017 is as shown in figure 31. Each bug currently in the BugHive as of March 14th 2017 is described in further detail below. Future bug additions are also outlined in section 5.3.1.

Bug_ID	Language	Variable/dependent	Variables	Generating	Regular Expression	Description	Wanted Cases	Extended Explanation	Solution/Explained	Fix	Commit Message	Num
1	Java	0	0	(Math.random())	((int)((Math.random() * 100)))	Error cast is higher priority unless wanted output of 0	result of this is always 0	first find what came after random() was it *100 ...	Consider changing this syntax to something like...	fixed casting error		1
2	Java	1	0	[string] == [string]	((w+s) != ((w+s) * s))	Refers to object reference rather than characters ...	[string] == null	object pointer is of the form 1c@E207U and gone...	need to use string.equals() to compare characters ...	[string].equals(string)	incorrect comparison of strings	2
3	Java	1	1	[int] = [int]++	((w+s) = s * (w+s)++)	assignment made before increment of second var...	[inc]++ is assigned equal to inc before inc...	increment the second variable before the assignment...	[int2]++ in [int] = [int2]	variable incremented after assignment changed		3
4	Java	1	0	return [int]++	((return) s + ((w/d) + 1) + 1)	useless increment of returned value	the line will terminate before the increment is ex...	increment the value before the return statement	[int]++ in return [int]	fixed increment return value statement		4
5	Java	1	0	[int] = [int]++	((w+s) = s * (w+s)++)	immediate overwrite of increment	the increment is immediately overwritten this new...	the line should be written as [int] = [int] + 1		fixed immediate overwrite of increment		5
6	Java	1	0	if (double == Double.NaN)	((w+s) = s * (w+s)++)	No value is ever equal to Double.NaN	this will always return false as no value will ever...	java uses a function Double.isNaN(double) to make this che...		fixed if(NaN) check		6
7	Java	1	0	if (boolean == boolean)	((if) s * ((w/d) + 1) * s)	assigning a boolean value rather than comparing in...	the statement will always return true, the condi...	to compare boolean values in a conditional stateme...	if (boolean) == (boolean)	fixed assignment rather than comparison with con...		7
8	Java	1	0	while (boolean) =	((while) s * ((w/d) + 1) * s)	assigning boolean variables rather than comparing	this leads to an infinite loop	"=" is used for comparing where as "=" is to ass...	while (boolean) = (boolean)	fixed bug assigning rather than comparing in whi...		8
9	Java	1	0	if (var1 == var2)	((if) s * ((w/d) + 1) * s)	Assigning variables in a conditional statement rat...	The statement will always return true	Use == to compare boolean values in a conditional ...	if (var1) == (var2)	fixed conditional statement assigning not comparin...		9
10	Java	1	0	double [var] = [int]	((double) s * ((w/d) + 1) * s)	This statement will always return an integer value ...	Dividing 12 in C is integer division and will res...	Dividing 10/2 will result in 0.5	double [var] = [int]	integer division changed to double division		10
11	Java	1	0	return [int]++	((return) s * ((w/d) + 1) * s)	useless increment of returned value	the value is returned before the increment occurs	increment the value before it, [int] = [int] + 1, in return [int]		incremented the value before its returned		11
12	Java	1	0	while (var1 == var2)	((while) s * ((w/d) + 1) * s)	Assigning variables in a conditional statement rat...	The statement will always return true	Use == to compare values in if (var1) == (var2)		fixed conditional statement assigning not comparin...		12

Figure 31 BugHive as of 14th March 2017

1. JAVA, CASTING MATH.RANDOM (J_CMR)

Overview: This bug constantly results in an integer value of 0 due to the casting priority. Although the bug can be correctly identified it cannot be explicitly fixed as the magnitude of integer desired remains unknown.

Variable Dependent: false **Variations:** false **Wanted Cases:** unless wanted output of 0.

General String: (int)Math.random()

Bug Description: Cast is higher priority than the Math.random function.

Extended Explanation: Math.random produces a double between 0 and 1, casting results in 0

Regular Expression: ((\\(int\\))|(\\(short\\))|(\\(long\\)))Math\\.random\\(\\)

Fix: //Consider changing this syntax to something like: (int)(Math.random()*x)

Solution Explained: Fix this bug by inserting a set of brackets around (Math.random()*x) and multiplying by x depending on the magnitude of integer required.

Commit Message: warning of casting error

2. JAVA, EQUALS COMPARING STRINGS (J_ECS)

Overview: This bug deals with the (assumed) incorrect comparison of two String variables. Using “==” to compare String variables does not compare the character values but instead the object pointers which are generally useless. BugBot makes the assumption that a developer does not wish to compare object pointers.

Variable Dependent: true **Variations:** false **Wanted Cases:** [string]==null

General String: [string]==[string]

Bug Description: Refers to object reference rather than characters of String

Extended Explanation: object pointer is of the form 1C@F267U and generally a useless check

Regular Expression: ((\\w+))s*((\\|=)|(\\!=))s*((\\w+))

Fix: [string].equals([string])

Solution Explained: str.equals(str) compares characters of a string and not object pointers

Commit Message: changed comparison of string pointers

3. JAVA, ASSIGNING INCREMENTED VARIABLES (J_AIV)

Overview: Identifying this bug also requires a level of assumption that the programmer intended [int1] to be equal to the incremented value of [int2], and not the un-incremented value followed by an increment of int2. This bug has the potential to induce false positives if directly fixed to increment [int2] before the assignment. For this reason, the bug was changed to simply comment a warning.

Variable Dependent: true

Variations: true

Wanted Cases: potentially

General String: [int1]=[int2]++

Bug Description: assignment is made before increment of second variable

Extended Explanation: i=inci++, i is assigned equal to inci before inci has been incremented, inci still increments.

Regular Expression: (\w+)\s*=\s*(\w+)\s*\++

Fix: //this integer is assigned equal to the unincremented value of the second integer, if this was unintentional please see the BugBotReport for a fix. (BugBotReport Fix: [int2]++; \n [int1]=[int2])

Solution Explained: increment the second variable before the assignment

Commit Message: variable incremented after assignment changed

4. JAVA, INCREMENT RETURNED VARIABLE (J_IRV)

Overview: This bug occurs when incrementing an integer in the return statement, the method will return before the integer is incremented. This bug is fixed by incrementing the integer before the return statement.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: return [int]++;

Bug Description: useless increment of returned value

Extended Explanation: the line will terminate before the increment is executed

Regular Expression: (return)\s+(\w+\d+)\s*\++;

Fix: [int]++; \n return [int];

Solution Explained: increment the value before the return statement

Commit Message: fixed increment return value statement

5. JAVA, IMMEDIATE OVERWRITE OF INCREMENT (J_IOI)

Overview: This bug is the incorrect attempt of incrementing an integer. The integer is immediately overwritten before it can be incremented and so the integer never increments.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: `[int]=[int]++`

Bug Description: immediate overwrite of increment

Extended Explanation: the increment is immediately overwritten thus never actually increments.

Regular Expression: `((\w+)\s*\s*=\s*(\w+)\s*\s*\s*)`

Fix: `[int]=[int]+1`

Solution Explained: the line should be written as `[int] = [int]+1`

Commit Message: fixed immediate overwrite of increment

6. JAVA, IS DOUBLE NOT A NUMBER (J_IDN)

Overview: This bug has proven common in preliminary testing. Used with the intention of checking if a double variable is equal to the Java undefined number Not a Number, this check will never return true as not even `Double.NaN` is equal to `Double.NaN`.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: `if([double]==Double.NaN)`

Bug Description: No value is ever equal to NaN

Extended Explanation: this will always return false as no value will ever be directly equal to `Double.NaN`

Regular Expression: `((\w+)\s*\s*=\s*(\w+)\s*\s*\s*(Double\.\NaN))`

Fix: `Double.isNaN([double])`

Solution Explained: java uses a function `Double.isNaN` to make this check

Commit Message: fixed `if(NaN)` check

7. JAVA, ASSIGNING BOOLEAN INSTEAD OF COMPARING (J_ABC)

Overview: This bug is the result of a missing “=” in a conditional statement. Assigning a boolean value in a conditional statement means the statement will always be true and the variable subject to change rather than check.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: if([boolean]=[boolean])

Bug Description: assigning a boolean value rather than comparing in an if or while statement

Extended Explanation: the statement will always return true, the conditional loop will always be entered

Regular Expression: ((if\(\)\s*((\w\d+)\s*(\={ 1 })\s*((\w\d+)\s*))\s*)

Fix: if([boolean]==[boolean])

Solution Explained: to compare boolean values in a conditional statement; use == or !=

Commit Message: fixed assignment rather than comparison within conditional statement

8. JAVA, WHILE LOOP ASSIGNING BOOLEAN INSTEAD OF COMPARING (J_WABC)

Overview: This bug is the result of a missing “=” in a conditional statement. Assigning a boolean value in a conditional statement means the statement will always be true and the variable subject to change rather than check.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: while([boolean]=[boolean])

Bug Description: assigning boolean variables rather than comparing

Extended Explanation: this leads to an infinite loop

Regular Expression: ((while\(\)\s*((\w\d+)\s*(\={ 1 })\s*((\w\d+)\s*))\s*)

Fix: while([boolean]==[boolean])

Solution Explained: fixed assigning rather than comparing in while loop

Commit Message: fixed increment return value statement

9. C, ASSIGNING VARIABLE INSTEAD OF COMPARING (C_AVC)

Overview: This bug is the result of a missing “=” in a conditional statement. Assigning a variable value in a conditional statement means the statement will always be true and the variable subject to change rather than check.

Variable Dependent: true

Variations: false

Wanted Cases: no

General String: if([var1]=[var2])

Bug Description: Assigning variables in a conditional statement rather than comparing them

Extended Explanation: The statement will always return true

Regular Expression: ((if\(\)\s*((\w\d+)\s*(\={1})\s*((\w\d+)\s*))\s*)

Fix: if([var1]==[var2])

Solution Explained: Use == to compare boolean values in a conditional statement

Commit Message: fixed conditional statement assigning not comparing

10. C, DOUBLES AND INTEGER DIVISION (C_DID)

Overview: If two integers are divided they result in an integer value, so if being assigned to a double variable it will be rounded down. The simple addition of a “.0” after the first integer value changes the result to double.

Variable Dependent: true

Variations: false

Wanted Cases: int [var]=[int]/[int]

General String: double [var]=[int]/[int]

Bug Description: This statement will always return an integer value even if assigned to a double variable

Extended Explanation: Dividing 1/2 in C is integer division and will result in 0 rather than 0.5

Regular Expression: (double\s*((\w+)\s*)\s*=\s*((\d+)\s*)\s*/\s*((\d+)\s*)

Fix: double [var]=[int].0/[int]

Solution Explained: Dividing 1.0/2 will result in 0.5

Commit Message: integer division changed to double-division

3.5 BUGBOT NAVIGATION

3.5.1 Autonomous Navigation: Web Crawling BitBucket

The navigation of the bot once deployed is autonomous unless specifically requested to scan elsewhere. This means that public remote repository users on BitBucket will be candidates for BugBot scanning. This autonomous scanning is carried out by periodical web crawling of BitBucket repositories when the queue reaches its lower threshold. Once BugBot reaches the threshold, random BitBucket repository URLs are added to the queue. The selection of these repositories is further discussed in section 3.3.8.1.

This aspect of BugBot's navigation is why it is essential BugBot does not induce false positives. It has not been requested to scan these repositories so it is imperative it provides a useful resource and service to the developers of the repository.

The autonomous navigation of the bot is anchored to BitBucket due to GitHub's ban of unauthorised bots. A Google search of BitBucket repositories returns over 65,000 results and so provides BugBot with ample candidate code.

3.5.2 Direct Request Users: Requested Navigation

Users particularly interested in the bot are able to make direct requests to their repository rather than waiting for the bot to get to their repository. BugBot both scans GitHub repositories for the inclusion of a BugBotFINDME.txt file in remote code repositories or it is provided with a user's URL to their repository through the BugBot website. From these requests BugBot would queue the requests and follow the standard bug scan protocol.

Direct request users are given priority in the FIFO queue and on average have to wait just half an hour for their repositories to be scanned.

3.5.2.1 Authorised Access of GitHub: BugBotFINDME.txt

The BugBotFINDME.txt file is a method of directly requesting BugBot scanning. The inclusion of this file means BugBot will repeatedly scan the repository after each commit. This is done by comparing the last time the repository was scanned with the time of the last commit. Extracting the time of the last commit is done online by inspecting the HTML of the repository homepage. This avoids the repeated scanning of a repository BugBot has already scanned and fixed but also offers users BugBot's continuous awareness of their repositories.

3.5.2.2 Direct Request to Repositories from Website

The direct requests made by users on the website allows users to simply paste the URL of their repository in a provided format. Once the user submits the URL, it is sent to the BugBot email which BugBot reads and queues.

BugBot does not check these repositories in the Records database so if a user wants their repository to be scanned more than once, this is possible.

3.6 ADMINISTRATOR GUI

3.6.1 Visual Basic using Visual Studio Implementation

The administrator's general user interface was developed using Visual Studio and was coded in Visual Basic. This offered a simple drag and drop platform to design the interfaces of the three different panels available to the administrator which were then supplemented with back end code to provide functionality. The interfaces are displayed and detailed from a user's perspective in sections 3.6.3 to 3.6.5 below.

The main panel consists of a web client that signs in to BugBot's Twitter account and displays the users feed, displays live statistics and progress monitoring by accessing the BugBotConfig file, functionality to run and reset BugBot and buttons to access the BugHive and Graphs panel. This is further detailed through the flowchart labelled figure 32 below.

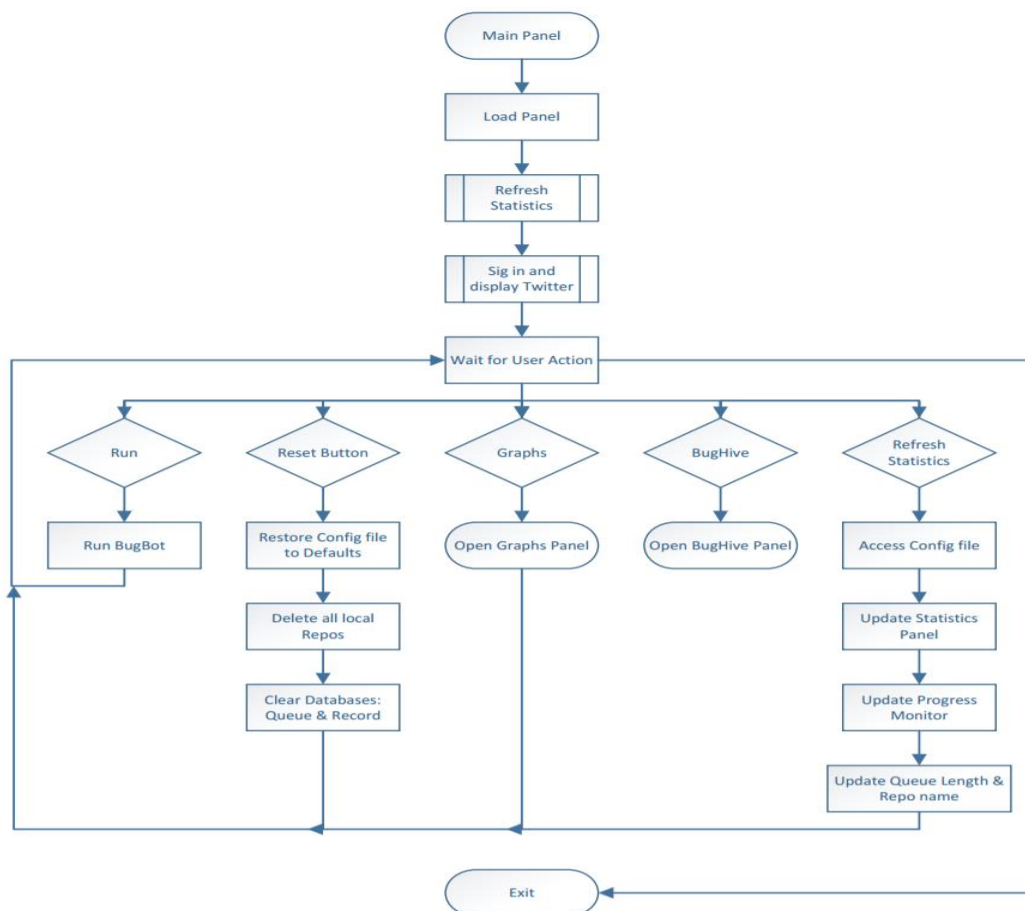


Figure 32 Main Panel of Administrators GUI Flowchart

The BugHive panel creates a connection to the BugHive database and displays each bug accordingly using the mySQL framework for Visual Basic. This panel displays all bugs in the BugHive clearly to the administrator by scrolling through the bugs using the Number text box. The code to execute this functionality is shown below in figure 33.

```
myDatabase = New MySqlConnection("server=danu6.it.nuigalway.ie;user id=
Dim strQuery As String = "SELECT * FROM BugHive WHERE Num=@Num"
Dim sqlCom As MySqlCommand = New MySqlCommand(strQuery, myDatabase)
With sqlCom
    .Parameters.AddWithValue("@Num", NumericUpDown1.Value)
End With
myDatabase.Open()
```

Figure 33 Visual Basic code for Connecting to BugHive Database

The BugHive panel allows for the easy addition, edit and deletion of bugs from the BugHive. Deleting from the BugHive leads to a dialog box ensuring the user knows a bug is about to be deleted, if the user clicks no; the method returns and no bug is deleted. If the user clicks yes the bug indexed by the number in the number text box is deleted from the BugHive. An short excerpt of the code to execute this functionality is shown in figure 34.

```
Dim strQuery As String = "DELETE FROM BugHive WHERE Num=@Num"
Dim sqlCom As MySqlCommand = New MySqlCommand(strQuery, myDatabase)
With sqlCom
    .Parameters.AddWithValue("@Num", NumericUpDown1.Value)
End With
Dim result As DialogResult = MsgBox("Are you sure you want to delete th
```

Figure 34 Visual Basic code to Delete a Bug from BugHive

The Graphs Panel accessed through the Graphs button displays two tabs for two different graph types; a bar chart and a pie chart representing the bugs found by bug id and the language distribution of the files scanned. The data for these graphs is accessed within the Refresh Statistic method, either on loading the panel or when the Refresh Statistics button is clicked on the Main Panel.

3.6.2 Administrator's Role

BugBot is essentially a web-crawling bot that operates using its own intuition, because of this it is necessary to consider the administrator's role in BugBot. The bot needs to be easily manageable and monitored. The BugBot GUI provides a mechanism to update and supervise the BugHive and bot progress without rewriting the source code or writing tedious SQL statements.

The GUI was made using Visual Studio and written in Visual Basic. It provides a connection to the BugHive database, an overview of the current statistics and progress of BugBot through a repository, and a web client with a connection set up to the BugBot Twitter Account on the main page, see figure 35.

3.6.3 Main Panel

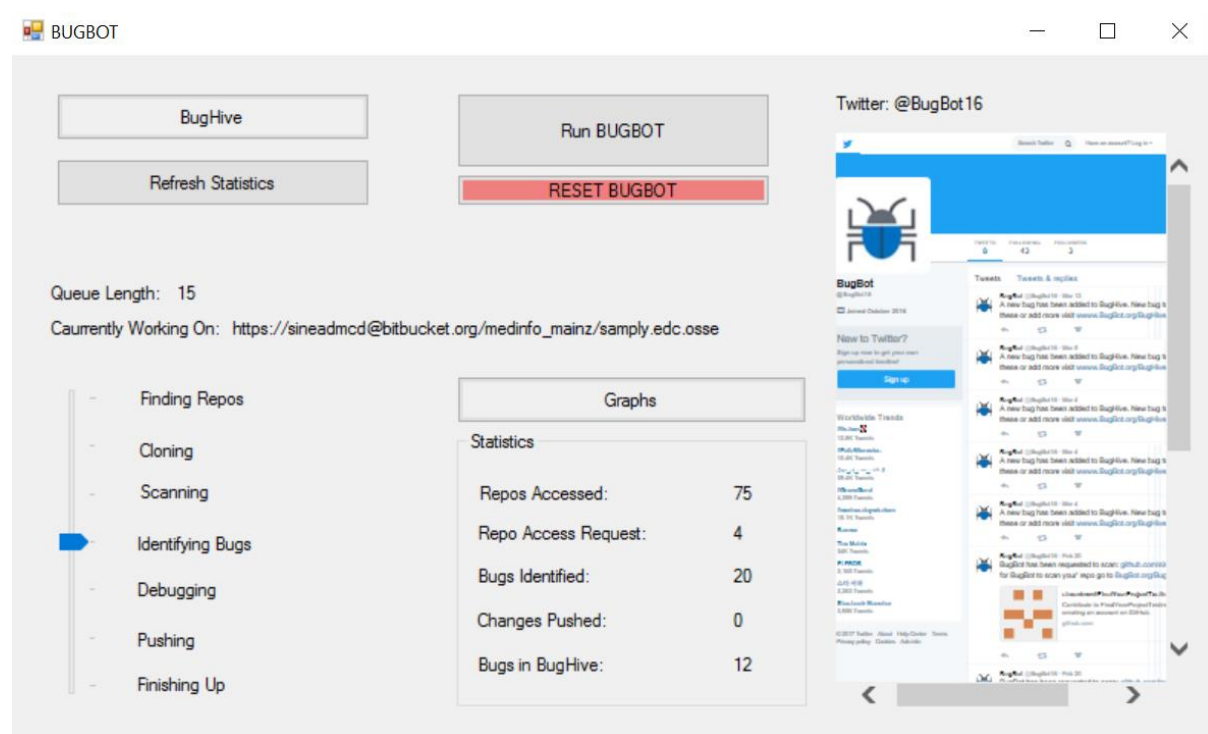


Figure 35 Administrator GUI- Main Panel

As BugBot continuously updates the BugBotConfig file as it is running, the GUI is a display of this data. By clicking “Refresh Statistics”, all statistics and progress monitors are updated. This includes the length of the queue, the repository BugBot is currently working on, the stage of the scan on the repository and general count statistics in the Statistics panel.

Both the “Run BUGBOT” button and “RESET” button were used mainly during the debugging and testing process. Run BugBot simply runs the Java program and the Reset button requires a password before resetting. It clears both the Queue and Record databases, deletes all the local clones present and restores the BugBotConfig file to its default values.

Both the BugHive and Graphs Button open other panels discussed in section 3.6.5 and 3.6.6. respectively.

3.6.4 BugHive Panel

Figure 36 Administrator GUI- Access to BugHive

The access to the BugHive allows easy addition, editing and deletion of bugs as well as a clear scroll through of the present bugs. By clicking the up and down arrows beside the Number text box all other text boxes change corresponding to the bugs “number” criteria as mentioned in section 3.4.1.

The delete button leads to an “Are you sure?” warning panel to ensure no bug is accidentally deleted. Because of BugBot’s dynamic source code bugs can be easily added and included in the next running scan if all criteria are inputted correctly.

3.6.5 Graphs Panel

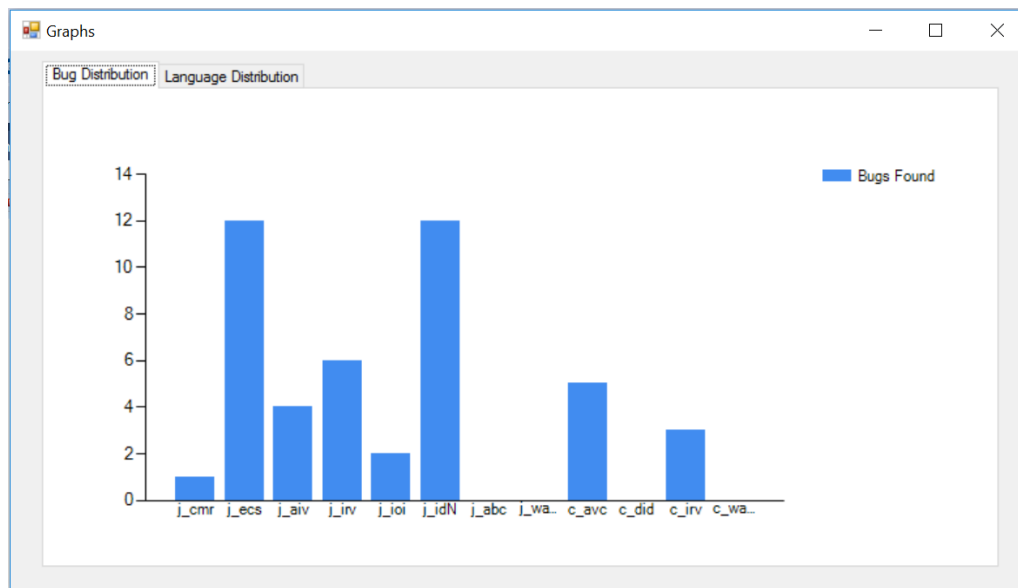


Figure 37 Administrator GUI Graph - Bug Frequency

The Graphs Panel is a further extension to the Administrators GUI's display of statistics, it provides easy and live access to the data BugBot saves in a DATA.csv file and BugBotConfig.txt file. The graphs shown are few of the statistical observations BugBot makes while scanning repositories. The two graphs available are the frequency of bugs found and language distribution of the files scanned. These graphs and the data they represent are further described in section 4.3.2.

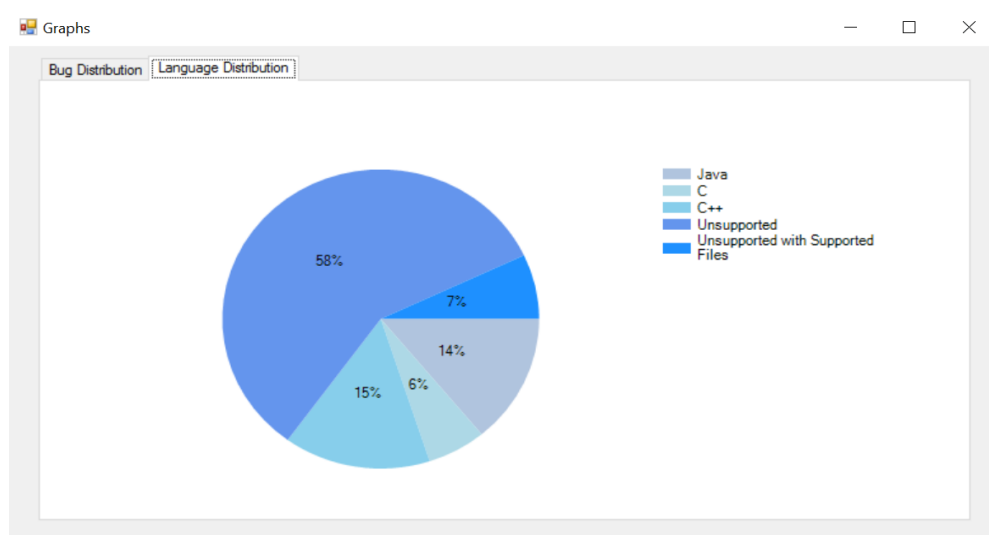


Figure 38 Administrator GUI Graph - Language Distribution

3.7 USER WEBSITE & TWITTER ACCOUNT

The user Website; www.BugBot.org is a WordPress developed website that offers a user-friendly approach for users to access information and have an input on the bugs present and future possibilities of bugs. It's three main functions are to provide information, allow for bug suggestions and additions, and the ability to make direct requests by simply pasting the URL of the repository. The published pages associated with BugBot.org are as follows:

- www.BugBot.org/Home
- www.BugBot.org/About
- www.BugBot.org/BugBot
- www.BugBot.org/BugHive
- www.BugBot.org/ContactUs
- www.BugBot.org/Request

Each of these pages offer different options and information.

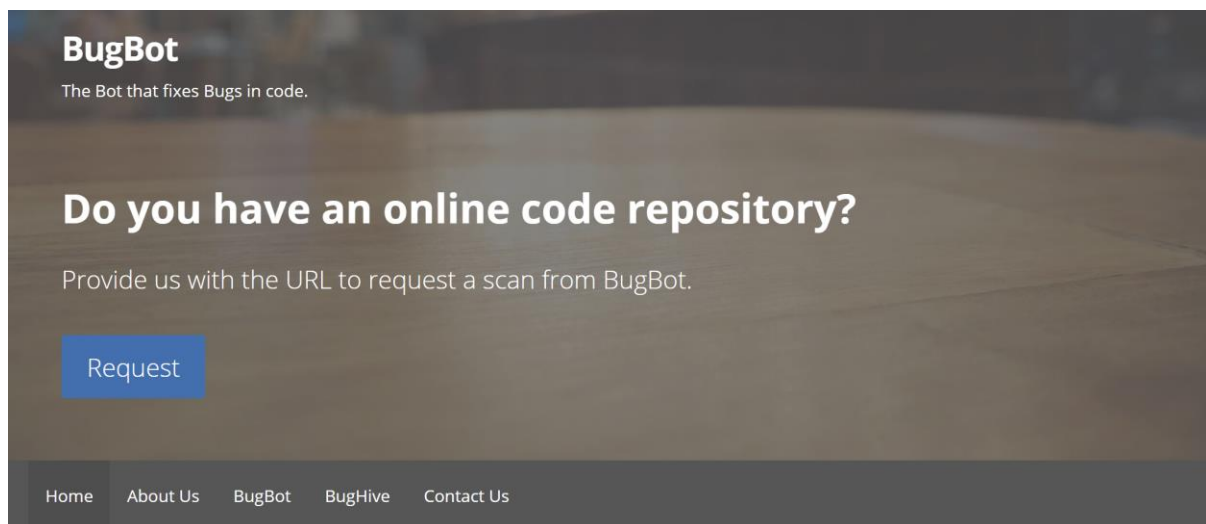


Figure 39 BugBot.org First Page Top Panel

3.7.1 User Information

The BugBot.org Homepage offers a short summary of the different features of BugBot in addition to links to all other pages. Each page includes the relevant information necessary for the user to fully understand the BugBot process. The “About Us” page provides information on the Aim, Concept and an “About Me” section. The BugBot tab provides a Direct Request Form and details on what providing BugBot with a Repository URL is allowing BugBot access to, and what they can expect. It also offers information on other BugBot navigation options such as the BugBotFINDME feature and its autonomous navigation. The BugHive tab offers users information on each deployed bug in a slideshow format and encourages users to comment on BugBot through the Contact Us form.

3.7.1.1 BugBot Twitter

Another method of providing users with information on BugBot is through the bot controlled Twitter account. This account is updated in the back-end code at certain mile-stones such as a bug being added to the BugHive accompanied by a link to the website to find more details on BugBot. It also tweets when BugBot has been directly requested to scan a repository and every time 100 repositories have been scanned. Examples of BugBot’s tweets are shown below in figure 40 and figure 41.



Figure 40 BugBot Tweet- 1000th Repository Accessed

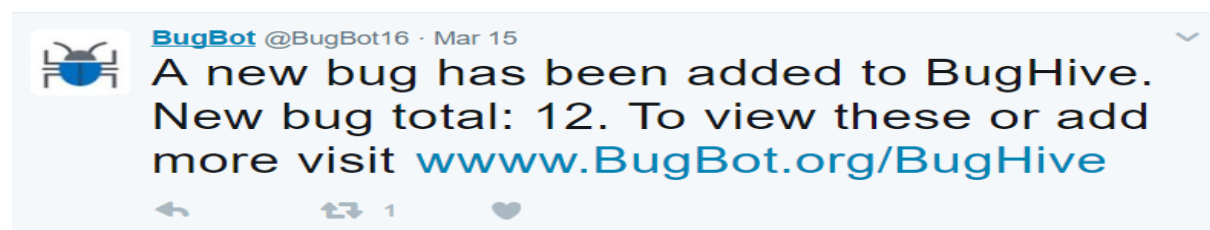


Figure 41 BugBot Tweet- 12th Bug Added to BugHive

3.7.2 Bug Addition

BugBot's aim is to help developers and so it is crucial BugBot offers its users a way of adding to BugBot regardless of their programming abilities. To fulfil this aim; the websites BugHive page, in addition to describing the current list of bugs it offers, provides users with a form with all the relevant criteria to fully define a bug.

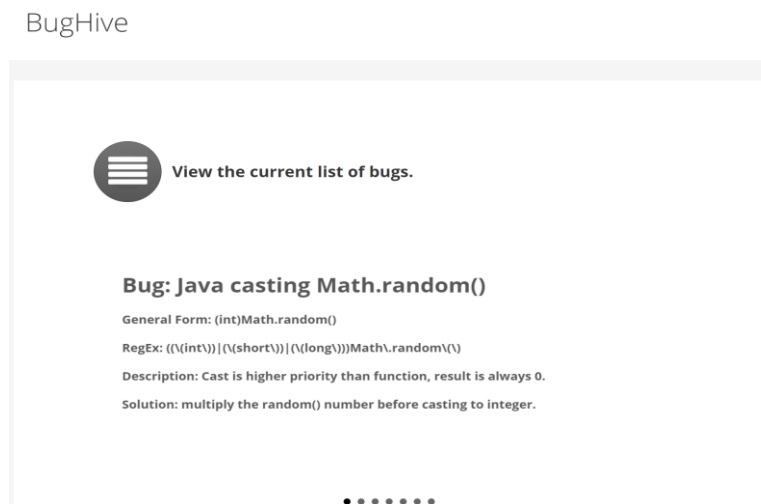


Figure 42 BugBot.org/BugHive Slideshow

Users can provide BugBot with as few or as many details of the bug as they can think of. Each criterion is thoroughly explained to further their knowledge on BugBot's process of finding bugs. Once a user submits the form, the form is sent to the administrator of the website for review. This, in addition to BugBot's open source code is a mechanism for BugBot's expansion.

3.7.3 Direct Requests

The Website also offers a form for users to submit their repository URL. This URL is automatically sent to BugBot's email address which BugBot periodically checks for updates. Users can enter this repository as many times as they wish as BugBot does not check the Records database for its previous entry.

Users are told about the various other ways BugBot can navigate to their repositories and exactly what they can expect if bugs are found i.e. a BugBotREPORT.txt file in their repository outlining what bugs were found as well as a history of commits made on each file that was "fixed".

Enter the Repository URL you would like BugBot to visit

Repository URL

`https://github.com/example/myrepo`

Ensure correct format. If BitBucket: `https://bitbucket/example/myrepo`

Submit

The Wordpress Plugin used for this mechanism is the Contact Forms plugin. Usually intended to automate the sending of “Contact Us” forms it was adapted to only send the required information with the required email title for BugBot to look for.

Currently BugBot only supports GitHub and BitBucket Repositories, users are informed of this and the required format of the URL is provided for clarity.

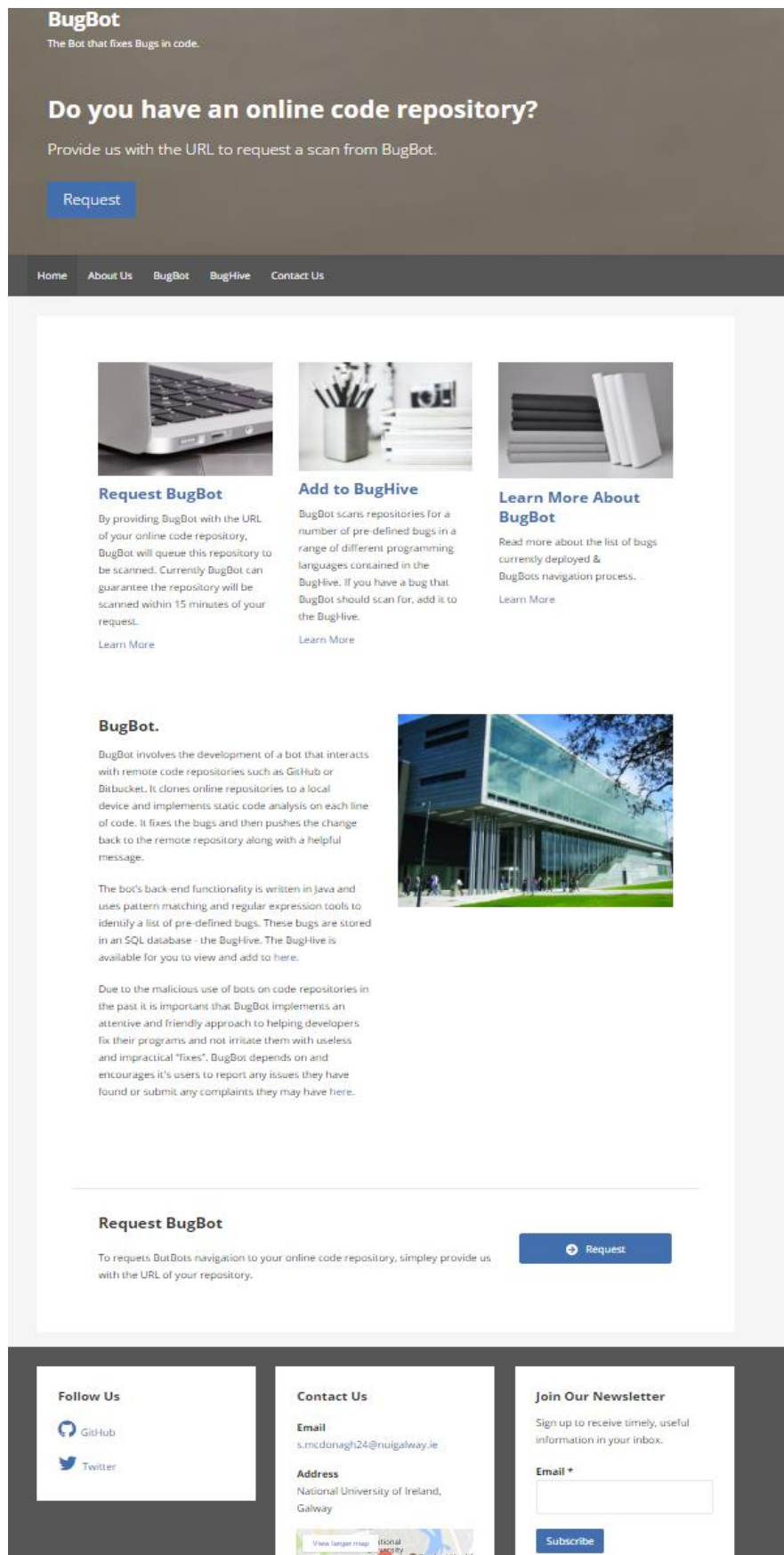


Figure 43 Webpage www.BugBot.org/Home

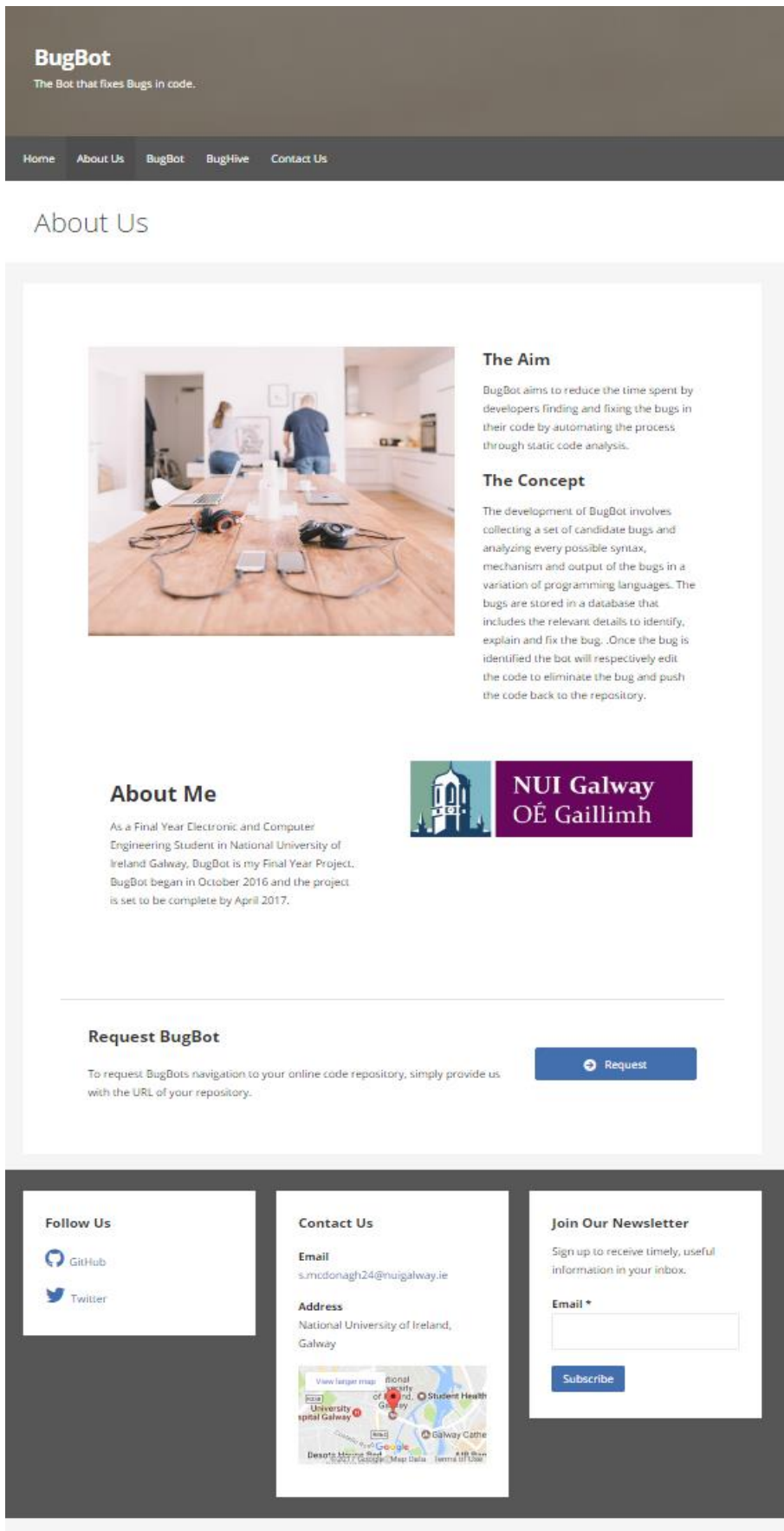


Figure 44 Webpage www.BugBot.org/About

BugBot
The Bot that fixes Bugs in code.

[Home](#)[About Us](#)[BugBot](#)[BugHive](#)[Contact Us](#)

BugBot

Enter the Repository URL you would like BugBot to visit

Repository URL

https://github.com/example/myrepo

Ensure correct format. If BitBucket: https://bitbucket/example/myrepo

Submit

What Now

Once you've pressed submit, your repository is sent to the queue to be scanned. The average wait time for BugBot to reach your repository is 15 minutes.

- If bugs were not present in your repo, expect nothing. BugBot cloned and scanned your repository and didn't find anything.
- If bugs were present in your repo, expect BugBot to make commits for each buggy file and a pull request with the changes. You will also receive a BugBotReport.txt file in the root of your repository with a summary of the bugs found.
 - Looking for more details of the bugs found? Click the "BugHive" tab above.

BugBot Navigation

Providing BugBot with the URL of your repository means the URL has now been added to the queue for scanning. The queue currently never exceeds 15 repositories so you should be visited shortly.

BugBotFINDME.txt

If your repository is on GitHub, you can ensure BugBot scans the repository each time you push new changes by adding a BugBotFINDME.txt file to the repository.

Autonomous Scanning

If your repository is on BitBucket, BugBot may have already visited your repository as it autonomously crawls through BitBucket repositories. If not, enter your repository URL above anyways!

Figure 45 Webpage www.BugBot.org/BugBot

86

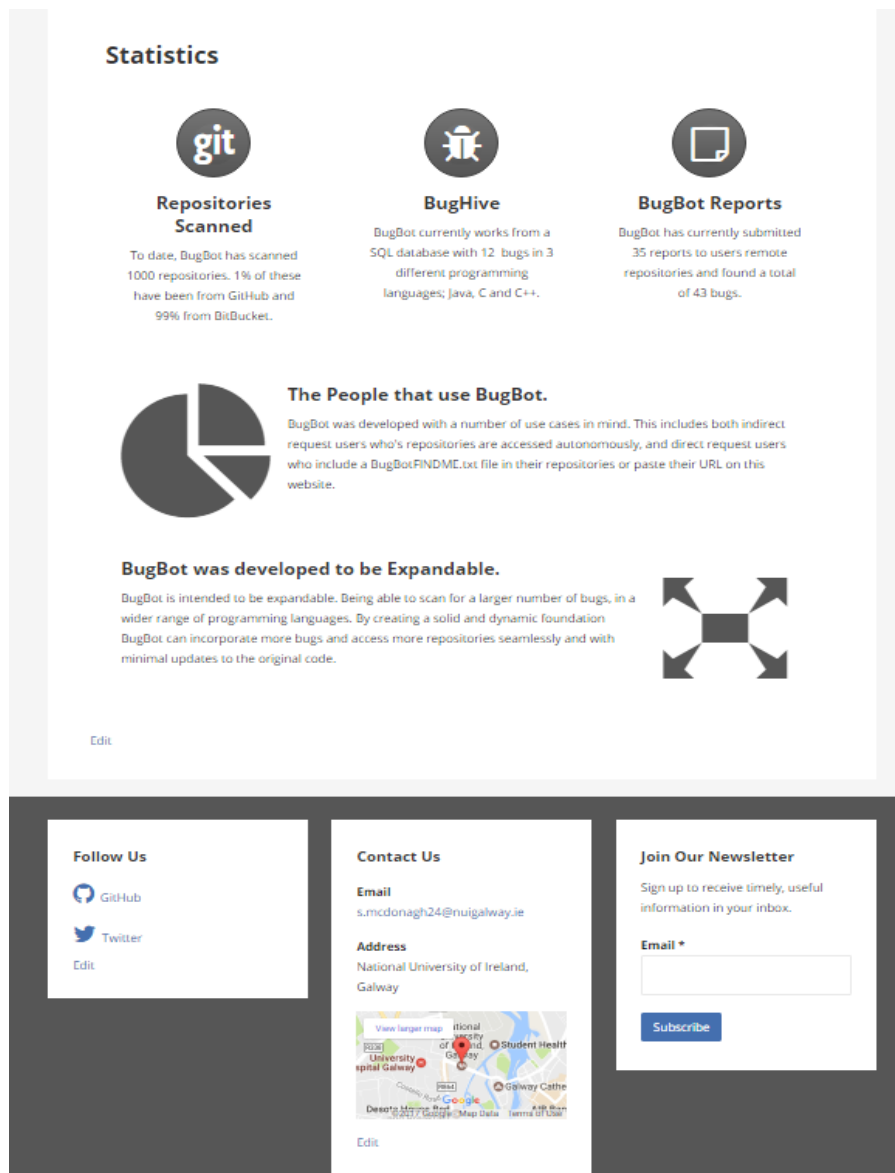


Figure 46 Webpage www.BugBot.org/BugBot

BugBot

The Bot that finds bugs in code.

[Home](#)
[About Us](#)
[BugBot](#)
[BugHive](#)
[Contact Us](#)

BugHive

View the current list of bugs.

Bug: Java is Double equal to NaN

General Form: `if([double]==Double.NaN)`

RegEx: `([^\w|\d]+)?[sS]*\v?[sS]*\v?(Double\.\NaN)`

Description: No value is ever equal to Double.NaN.

Solution: Java uses a function `Double.isNaN` to make this check.

Fix: `Double.isNaN([double])`

If you have a suggestion for a bug, add it to BugHive.

Bug Suggestion

First Name

Email

Language [?]

☐ Variable Dependent [?]

General Form [?]

Regular Expression Form [?]

If unfamiliar with java's regex leave this blank and provide a detailed description!

Description

Wanted Cases [?]

Extended Explanation [?]

Solution Explanation [?]

The Fix [?]

Commit Message [?]

Submit

Figure 47 Webpage www.BugBot.org/BugHive

CHAPTER 4 RESULTS

4.1 TESTING

Testing is a crucial component to the success of BugBot. For a bot to make assumptions on human written code it must have a solid basis to make a judgement and subsequently change the code.

To ensure this was the case, each bug was extensively tested against its regular expression definition before being deployed in the official BugHive. In addition to this testing, BugBot was monitored on a probationary period whereby it would clone and scan repositories as intended, but would not alert the owners of the repository in any way by forking, pushing or making a pull requests.

Precautionary measures must be taken to ensure over-all robustness of the system. This involves exploring and researching any possible issues that can arise during each basic step of the BugBot - pull, fix and push and so various tests of the back-end code were conducted and reviewed at each step of the development. This was mainly done by printing to the console as a repository progressed through the different classes and methods. The administrator GUI was also a key component of this testing as statistics and location were continually updated.

4.1.1 Testing Regular Expression Patterns

To ensure the regular expression patterns of bugs were correct, a separate Testing project was created on Eclipse [56]. This allowed a console input of a regular expression pattern and a test String to match with the pattern. Both the program used for testing and the subsequent console output of this program are shown in figure 48 and 49 respectively.

```

System.out.println("enter regex: ");

Pattern pattern =
Pattern.compile(in.nextLine());

System.out.println("enter string ");
Matcher matcher =
pattern.matcher(in.nextLine());

boolean found = false;
while (matcher.find()) {
    System.out.println("I found the text '" +
        matcher.group()+ "' starting at " + matcher.start() +
        "index,and ending at index " + matcher.end());
    found = true;
}
if(!found){
    System.out.println("No match found");
}

```

Figure 48 Regex Testing Source Code

```

enter regex:
((\w+)\s*((\=\=)|(\!\=))\s*((\w)+)
enter string
if(string1==string2){
I found the text 'string1==string2' starting at 3index,and ending at index 19
enter regex:
((\w+)\s*((\=\=)|(\!\=))\s*((\w)+)
enter string
if(string1!=string2){
I found the text 'string1!=string2' starting at 3index,and ending at index 19
enter regex:

```

Figure 49 Console Output of Regex Test Program

All information of Java's Regular Expression framework was acquired form Oracles regex online tutorial [57].

The creation of the regular expression definitions for each bug required extensive analysis of the following:

- All possibilities of how the bug can present itself in context.
- Wanted cases of the bug that would disallow immediate editing of the code.
- Exceptions and limitations to defining the bug.

Each bug defined as a result of this process is defined and scannable in the BugHive.

4.1.2 BugBot Probation

While BugBot is running on NetBeans, certain details are printed to the console to track its progress. This was originally used for debugging purposes but currently gives live insight into the progress through the scanning of a repository. During BugBot's probationary period the console outputs were constantly analysed to check what files it was scanning, bugs it was identifying and how it was fixing them. This highlighted three major issues:

1. **Casting Limitations:** BugBot does not yet have the capability to recognise casted variables and variables subject to toString() methods.
2. **Null as a String:** BugBot initialized null as a variable of type String.
3. **Duplicate Regex Definitions:** BugBot had two identical regular expression definitions for two different types of bugs.

Casting Limitations

BugBot does not yet have the ability to recognise when variables are casted to different variable types meant that a previously defined bug had to be removed until this capability was added. Originally the bug: [var].equals([var]) whereby both mentioned variables were of different variable types was included in the BugHive and its unique definition catered to in the source code.

However, this implementation had to be removed as it was observed that 100% of the identifications of this bug were false positives. Consider the following:

```
int i=0;
String str1 ="0";
if(str1.equals(Integer.toString(i)))
{
    // BugBot identifide this as a bug
}
```

Figure 50 BugBot's Incorrect Identification of a Bug

BugBot is unable to identify that the integer is not a String and not of a different variable type.

Null as a String

BugBot initialized null as a variable of type String and stored this in the variable array for later parsing. This would seem standard as many String variables are given a null value. However, this resulted in over 28 false positives over the course of just under 100 repository scans. The unforeseen reason for this being the defined bug: `[string]==[string]` has an exception of checking if a string is equal to null. Consider the following:

```
String str1 = "hello";
if(str1== null){
    // BugBot identifies this as a bug
    // BugBot attempts to fix the "bug" by rewriting as:
    if(str1.equals(null))
    {
    }
}
```

Object equals "null" is never true

(Alt-Enter shows hints)

Figure 51 BugBot Incorrect Identification of a Bug

BugBot was actually replacing functioning code with bugs that the compiler points out to programmers. The solution to this was to simply remove the initialisation of null as a string variable.

Duplicate Regex Definitions

The final major issue that was highlighted during BugBot probation was that there exists two different bugs within BugHive with the exact same regular expression definitions. This is when the “Variations” criterion was added to BugHive.

The two bugs are outlined as follows in their General Form and corresponding Descriptions:

`[int1]=[int2]++;` Assignment is made before increment of second variable is executed.

`[int]=[int]++;` Assignment is made before increment of itself is carried out.

The difference between these two bugs is that the first contains two variables and the second contains only one variable, two times. The regular expression definition for both bugs is:

`(\w+)\s*\s*=\s*(\w+)\s*\s*++`

However, (\w+) stands for different variables in the second mentioned bug. This error resulted in BugBot only identifying the first mentioned bug regardless of whether there were two different variables in the bug. Consider:

```
int i=2;
int j=3;

i=i++; // i remains 2: BUG 1
/** BUGBOT FIXES ****/
i=i+1; // i is equal to 3 as expected for Fix 1

i=j++; // i is equal to 3 and not 4: BUG 2
/** BUGBOT FIXES ****/
i=i+1; // i is equal to 3, not expected as programmer intended increment of j for Fix 2
      // i=j+1; is the expected fix.
```

Figure 52 BugBot Incorrect Fix of a Bug

This issue was resolved by adding the Variations criterion to the BugHive, if BugBot were to identify a bug with Variations set to '1' it would have to conduct further analysis to ensure it's identifying the correct bug i.e. in this case checking whether the bug contained one or two variables.

The bug outlined in figure 52: `i=j++` has since been changed to simply comment the suggested fix, as further research revealed that often times this “bug” is intentional and not in fact a bug as defined by static analysers FindBugs and Coverity.

4.2 BUGBOT DATA RECORDING

While BugBot is scanning repositories it is simultaneously monitoring performance and collecting data on each repository scanned requiring constant update of external data files, namely the BugBotConfig text file and the DATA.csv file. This, in addition to the BugBotReport file committed to each repository is what allowed useful statistics and performance measures to be gathered and analysed.

4.2.1 BugBotReport.txt

The BugBotReport file was committed to each repository with a list of the bugs the repository contained in addition to a short summary of each bug and a link to the BugBot.org website for further information. An example of a BugBotReport file is shown below in figure 53.

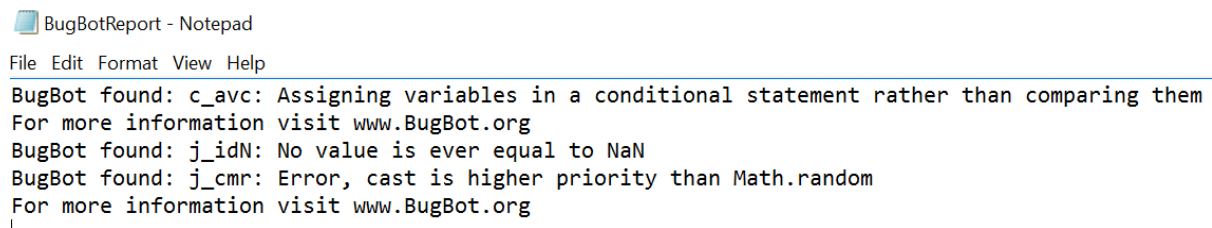


Figure 53 BugBot Report as Committed to a Repository

The reports were generated by adding a line of text to an `ArrayList<String>` each time a bug was present with the corresponding bug Description of the bug from the BugHive.

4.2.2 DATA.csv

This file collects the unique data of each scanned repository. It contains the following information in the following order:

- The URL of the Repository.
- The Language listed in the metadata of the repository.
- The number of Java files the repository contained.
- The number of C files the repository contained (including C++).
- The number of declared variables in the repository (limited to Java).

- The Lines of Code in the repository.
- The Time taken to scan the repository, from clone to finish.
- The number of Bugs identified in the repository.

An excerpt of the DATA.csv file is shown below in figure 54.

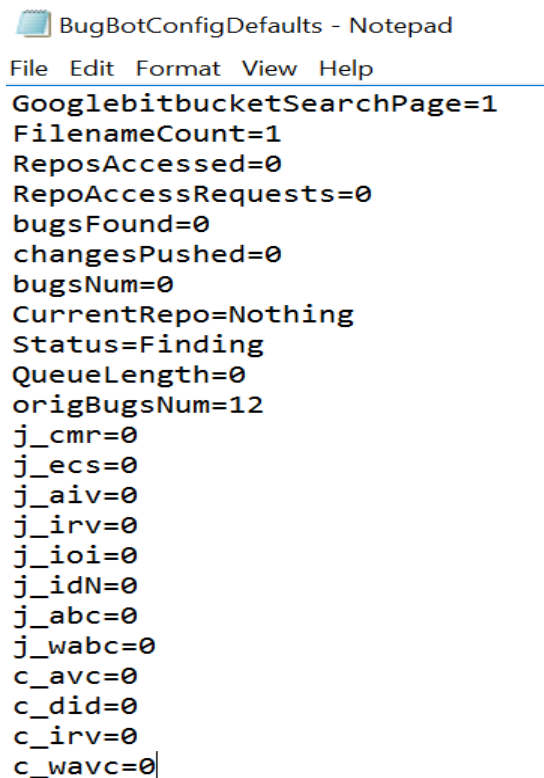
```
https://BugBot16@bitbucket.org/berkeleylab/metabat,C++,0javaF,3cF,0V,2983L,0:02:07.046,0B,
https://BugBot16@bitbucket.org/apurde/svg-out,JAVA,19javaF,0cF,1162V,2407L,0:01:31.319,0B,
https://BugBot16@bitbucket.org/atlassian/docker-atlassian-bitbucket-server,unsupported,0javaF,0cF,0V,0L,0:00:03.634,0B,
https://BugBot16@bitbucket.org/siani/javafmi,JAVA,260javaF,23cF,126114V,27805L,0:22:58.179,10B,
```

Figure 54 DATA.csv File

This data is analysed and displayed in a number of graphs in section 4.3.2.

4.2.3 BugBotConfig.txt

The BugBotConfig file contains most of the information that the administration GUI requires to provide live monitoring of BugBot's progress. The file consists of the following measures that are updated frequently throughout the scanning of a singular repository.



```
BugBotConfigDefaults - Notepad
File Edit Format View Help
GooglebitbucketSearchPage=1
FilenameCount=1
ReposAccessed=0
RepoAccessRequests=0
bugsFound=0
changesPushed=0
bugsNum=0
CurrentRepo=Nothing
Status=Finding
QueueLength=0
origBugsNum=12
j_cmr=0
j_ecs=0
j_aiv=0
j_irv=0
j_ioi=0
j_idN=0
j_abc=0
j_wabc=0
c_avc=0
c_did=0
c_irv=0
c_wavc=0
```

Figure 55 BugBotConfig Default Values

4.3 PERFORMANCE AND STATISTICS

4.3.1 Successful Identification & Fix Case Study

The following is an example of a repository that contained a total of ten bugs, all of the same type.

The bug referred to as j_idN is a Java bug which concerns the incorrect check of a number equal to Java's Double.NaN. Java's "Not A Number" is the result of a floating-point operation with an incorrect input parameter that causes the operation to produce some undefined result [58]. Often times developers check if a variable is equal to this value by typing something like:

```
if(d==Double.NaN){
```

This check will always return false as no number is ever equal to this value, not even itself. Instead developers should use the Java function:

```
If(Double.isNaN(d)){
```

The repository: <https://bitbucket.org/siani/javafmi> had ten total occurrences of this bug distributed between two files. This repository is a set of components to work with the Functional Mock-up Interface FMI. It is listed as a Java repository in its metadata and contains 260 Java files, 23 C files 27,805 lines of code and took approximately 15 minutes to scan. These were the details printed to the DATA.csv document.

In figure 56 we can see the live console output as BugBot identified and respectively fixed the bug it was consistently finding and correctly fixing. After fixing all bugs in the file it committed the file with the message: fixed if(Double.NaN) check.

After committing all fixes of the bug across two separate files, BugBot also included a BugBotREPORT.txt file as a separate commit outlining the bug description and bug fix as shown in Figure 58.

Files changed (1)

+4 -0 A BugBotREPORT.txt

```

BugBotREPORT.txt  ADDED
1  +BugBot found: j_idN: No value is ever equal to Double.NaN
2  +   BUG: if(min==Double.NaN){
3  +   FIX: if(Double.isNaN(min)){
4  +For more information visit www.BugBot.org

```

Figure 58 The Commit of the BugBotREPORT.txt file

BugBot then issued a pull request to the master repository. The pull request is currently open but the repository owner is expected to merge these fixes.

Commits

All branches ▾				Find commits
Author	Commit	Message	Date	
BugBot	a0ec06e	BugBot fixed the following	10 hours ago	
BugBot	2d94101	fixed if(Double.NaN) check	10 hours ago	
BugBot	9e43308	fixed if(Double.NaN) check	10 hours ago	
Raúl López	08a7756	Change maven distribution to siani.	2017-03-03	
Jose Evora	8fefa9d	fmiVersion detection fixed when attribute "version" is present. Is...	wrapper_2.20.2 develop	2017-03-02
Jose Evora	94dc9ba	fixed issue #76	wrapper_2.20.1 develop	2017-03-01

Figure 59 The Current Network Graph.

4.3.2 Repository Scanning & Bug Finding Statistics

From BugBot's continuous data recording as described in section 4.2 the following graphs have been created and analysed. As of 28th March 2017 BugBot had scanned 1008 repositories from both GitHub and BitBucket. From these 1008 repositories 42 bugs were found across 11 different repositories.

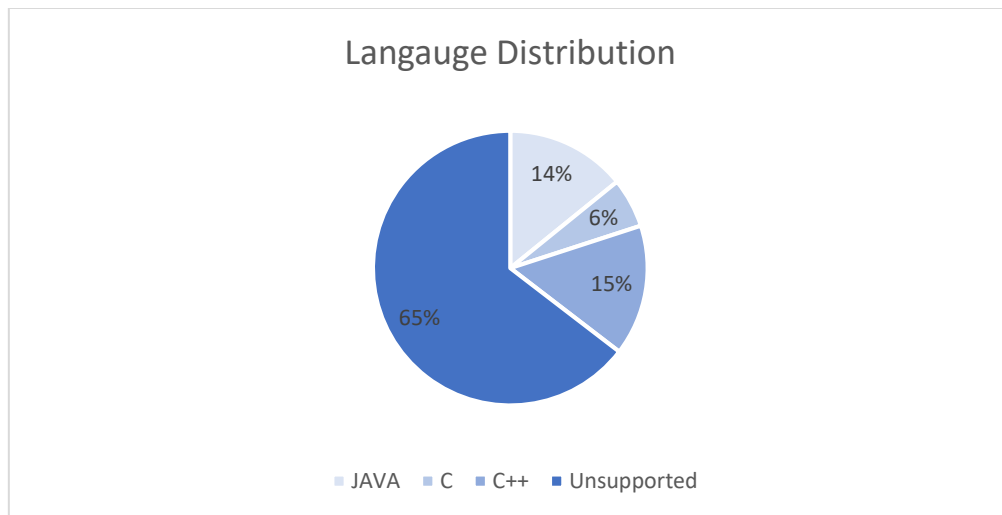


Figure 60 Language Distribution among Cloned BitBucket Repositories

When cloning BitBucket repositories the metadata of the repository is extracted from the remote repository page, figure 60 represents these languages listed in this metadata. Note that although 65% of the repositories cloned were not labelled as Java, C or C++, 10% of the unsupported language repositories contained either Java, C or C++ files. The most notable example of which was labelled as an unsupported language (Fortran) but in fact contained 116 C files. As shown in figure 61, a total of 58% of cloned repositories were cloned but not scanned as they contained no Java, C or C++ files.

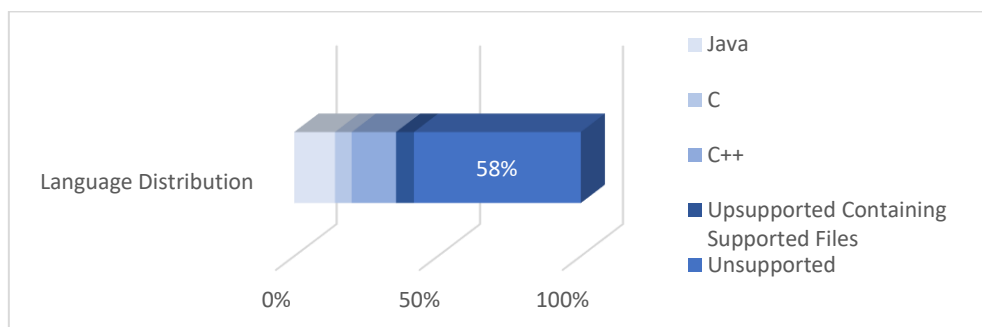


Figure 61 Files Cloned & Not Scanned

Figure 62 shows the time taken to clone and scan a repository vs the number of lines of code. The outlier at 1 hour and 26 minutes with 0 lines of code was a particularly large file size with 0 lines of scannable code i.e. code of an unsupported language. Of the 1000 repositories scanned, this repository provided the only justification for adapting the approach of only cloning repositories that list a supported programming language in the metadata.

All other repositories show a relatively linear trend between time taken and lines of code. The largest file scanned was over 350,000 lines of Java code, 1847 Java files, and took 3 hours and 16 minutes to clone and scan. This repository contained 2 bugs, both the java, increment returned variable bug.

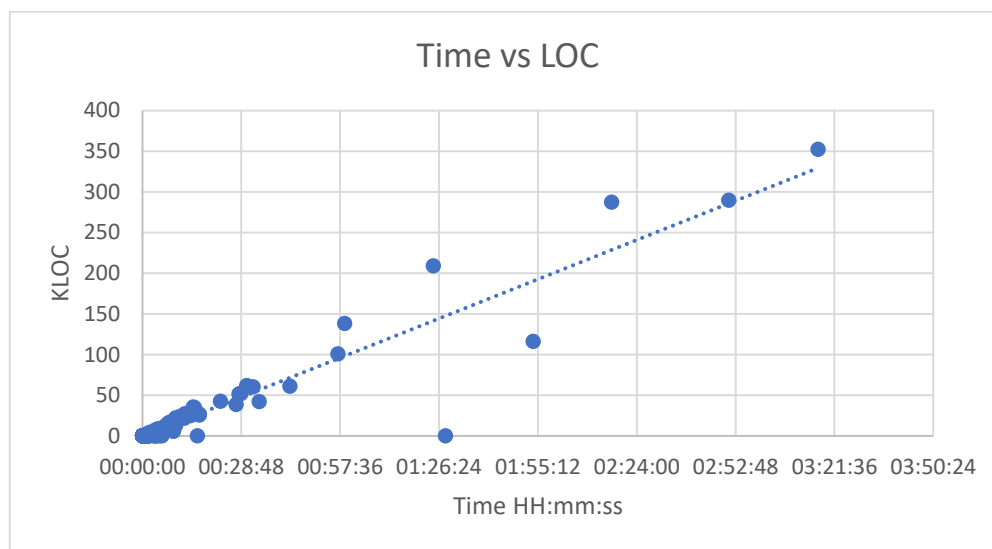


Figure 62 Time to Scan vs Lines of Code

In figure 62, kilo-lines of code (KLOC) is graphed against number of files. The repositories in which bugs were present are highlighted in red. It is evident from this graph that the larger the repository in terms of lines of code and number of files, the more likely they were to contain a bug. However, the number of bugs present did not correlate to repository size. As the three largest projects, in terms of LOC, included in this graph contained either one or two bugs, the three smallest bug-present projects contained two, ten and eight bugs.

It could have been expected that the repositories with less files and more lines of code (points in the upper left hand corner of the graph) would contain more bugs due to a lack of modularity⁸. However, in general, this is not evident from these results.

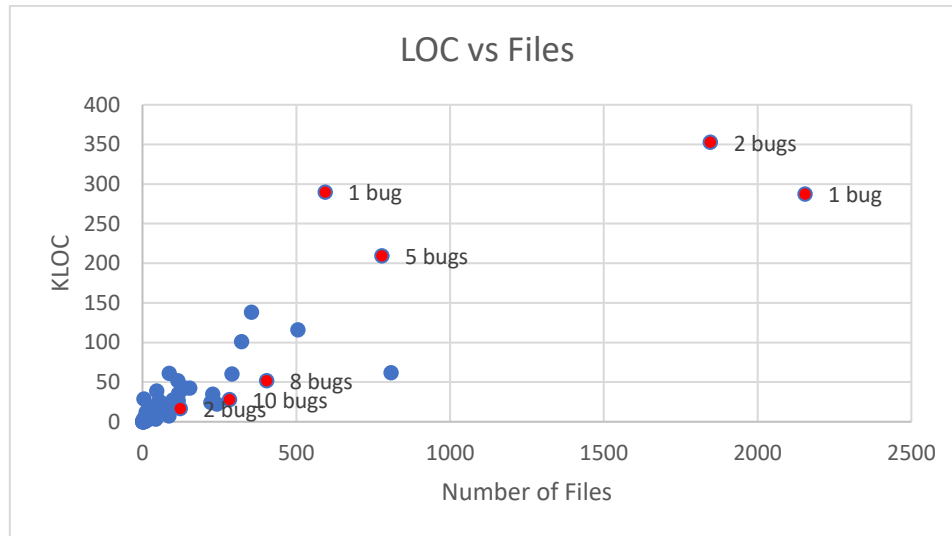


Figure 63 Lines of Code vs File Size

In figure 63, seven repositories in which bugs were present are graphed as the number of files vs the number of bugs found for further analysis. Surprisingly, the number of bugs found in a repository had absolutely no positive correlation to the number of files.

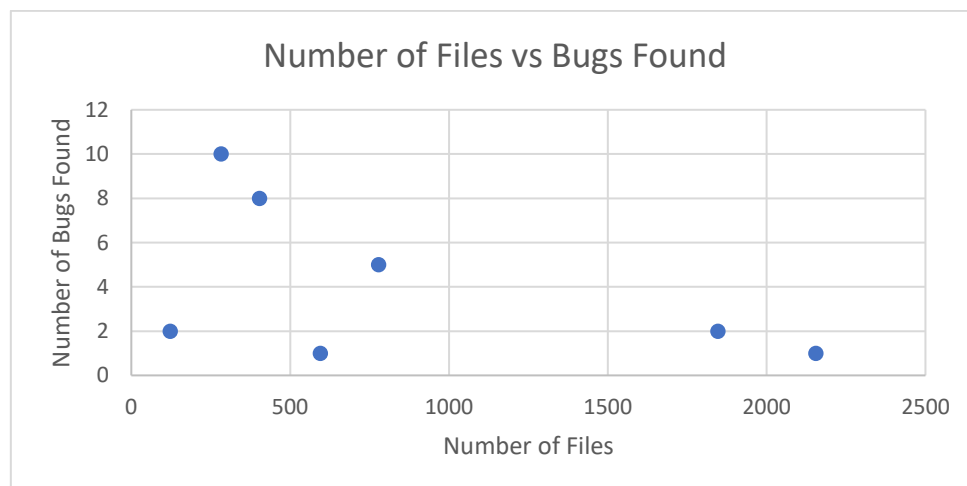


Figure 64 Number of Files vs Bugs Found

⁸ Modularity is the measure of separate and independent functions of code. Generally high modularity is recommended for low cyclomatic complexity.

Averages		
	Overall	Buggy Repo
Average Number of Files	176	396
Average Number of LOC	38,514	147,622
Average Time to Scan	00:14:32	01:25:17

Table 5 Average Figures of Buggy Repositories

The averages shown in table 5 above further reflect the fact that the larger repositories are prone to bugs as the average number of files in the cloned repositories was 176 and the average in a “buggy” repository was 396. Both the lines of code and time to scan variables reflect the same fact.

As of the 28th March 2017, the total number of bugs found was 43. The different bugs that were found are displayed in figure 65. The most commonly found bug is the Java check for a variable equal to Double.NaN. This bug was found a total of 12 times across two repositories. The process of identifying and fixing this bug is described in detail in section 4.3.1.

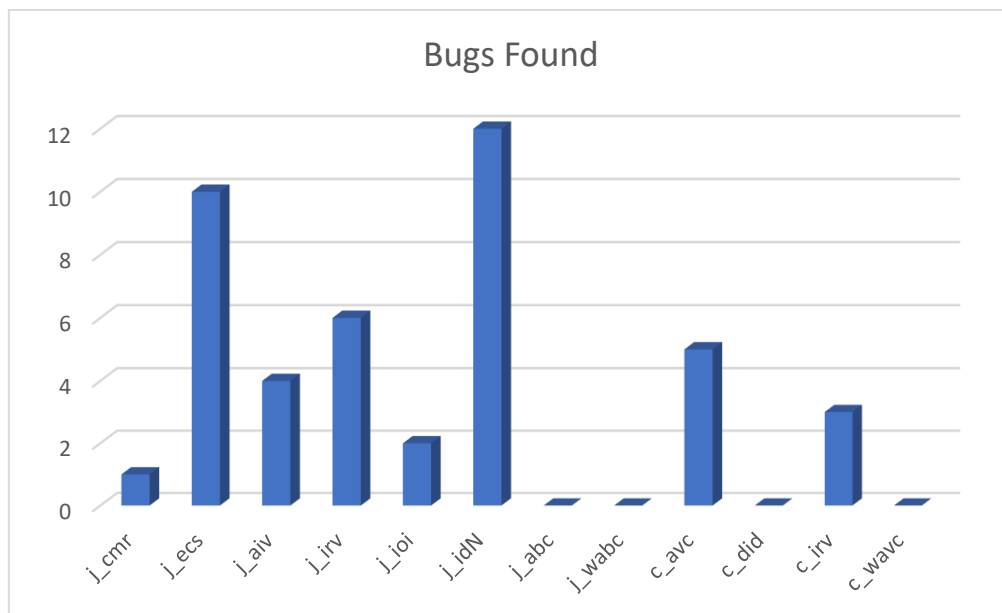


Figure 65 Bug Distribution

Figure 66 is a further breakdown of the bugs found by language. 81% (35) of the bugs found were Java and 19% (7) were C bugs found in either C++ and C code.

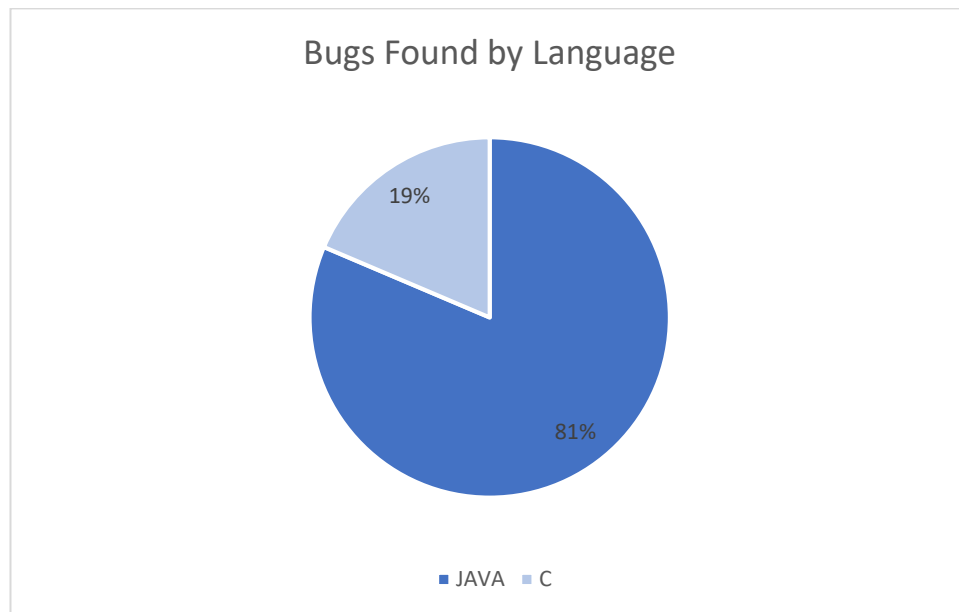


Figure 66 Bugs Found by Language

As the number of Java bugs in the BugHive outnumber the C bugs 8:4 it is useful to normalise this data to identify the “buggier” language.

$$35 \text{ [Java bugs found]} / 8 \text{ [Java Bugs]} = 4.375 \text{ bugs per definition}$$

$$8 \text{ [C bugs found]} / 4 \text{ [C bugs]} = 2 \text{ bugs per definition}$$

$$\text{Normalised Rate of Java bugs} = 4.375 / 6.375 = 68.6\%$$

$$\text{Normalised Rate of C bugs} = 2 / 6.375 = 31.4\%$$

These figures are an estimate of the bugs that would be found by language if there was an equal number of Java and C bugs in the BugHive.

Repositories with Multiples of the same Bug							
URL	Java Files	C Files	LOC	Time	Bugs	Bug Types	Repeat Bug
https://BugBot16@bitbucket.org/connect2id/nimbus-jose-jwt	291	0	60,258	00:33:54	5	2j_irv,3j_ecs	✓
https://BugBot16@bitbucket.org/axelclk/info.bliki.wiki	404	0	51,733	00:28:09	8	1j_irv, 7j_ecs	✓
https://BugBot16@bitbucket.org/jmarsden/jsonij	123	0	16,489	00:07:45	2	2j_idN	✓
https://BugBot16@bitbucket.org/l2jsrver/l2j_server	1847	0	352,424	03:16:47	2	2j_irv	✓
https://BugBot16@bitbucket.org/tasseladmin/tassel-5-source	778	1	209,246	01:24:41	5	1j_irv, 4j_aiv	✓
https://BugBot16@bitbucket.org/siani/javafmi	260	23	27,805	00:15:52	10	10j_idN	✓

Table 6 Repositories Repeating the Same Bug

The data shown in table 6 is collected in the DATA.csv file discussed in section 4.2.2. A pattern BugBot observed while scanning repositories is that bugs are identified in groups; unsurprisingly, if a bug is present in a project, often times the same bug is repeated again in the project, more than likely the bugs were created by the same programmer who was unaware of the mistake being made. The most notable instance of this is the same bug being repeated ten times throughout one repository.

Direct Request Wait Times			
Repo	Requested	Scanned	Wait Time
1	10:27	10:43	00:16
2	10:56	14:38	03:42
3	14:31	14:39	00:08
4	15:04	15:17	00:13
5	15:28	16:53	01:25
6	15:29	16:54	01:25
7	17:05	17:12	00:07
8	17:23	17:49	00:26
9	19:42	20:01	00:19
10	19:50	19:52	00:02
Average:			00:48

Table 7 Direct Request Wait Times

From a sample of 10 direct requests made from the website, the average wait time experienced by users for their repositories to be scanned was 48 minutes. The maximum wait time experienced was 3 hours and 42 minutes, this was due to being added to the queue shortly after a clone of a large repository had started. The minimum wait time experienced was 2 minutes, this was due to being added to the queue a mere two minutes before the leading repository was finished scanning. For now, the wait times experienced by users is unpredictable and the only method of speeding up the wait time

is shortening the minimum queue length threshold⁹ as discussed in section 3.3.2.1. Even so, the wait time relies entirely on the unpredictable size of the repositories already in the queue.

4.3.3 Discussion of Findings

From the data presented above it can be concluded that although the larger projects are more likely to contain a bug, the size of the project does not influence the number of bugs found. In fact, the size of the project and number of bugs found in bug-present repositories could be seen to have a negative correlation. This could be related to the study aforementioned by Coverity Scan which concluded that open source code was of higher quality and contained fewer details than proprietary code as a result of the high number of people contributing and viewing the code [39].

Java proved the predominately buggier language. Despite the number of Java files scanned being lower than the number of C and C++ files scanned (figure 60), the number of Java bugs found remained higher than the number of C bugs found after normalising the figures to represent an equal number of bugs defined in the BugHive.

Another interesting observation made by BugBot is that the probability of a project containing the same bug more than once is higher than the project containing one bug; as three bug-present repositories contained just one bug and eight contained the same bug at least twice (as of the 28th March 2017). This may not be surprising as a programmer is obviously liable to make the same mistake twice if they are unaware of the error. However, this observation does further strengthen the argument that an automated static analysis tool for remote repositories is necessary not only to fix mistakes but also educating programmers on commonly made errors.

⁹ The minimum Queue length threshold variable was set to 10 at the time of testing.

CHAPTER 5 CONCLUSIONS

5.1 CHALLENGES

5.1.1 Unauthorised Bot Access

GitHub bans the use of unauthorised bots since 2012. This was originally perceived detrimental as BugBot was intended to anchor itself to GitHub and crawl through the public repositories hosted there. Brian Doll, the marketing head of GitHub had good intentions when deciding to ban unauthorised bots, saying “We don’t want to be the platform where you don’t want to run a project because you’re going to be accosted by so much (bot) activity” [59].

The rule was respected and in overcoming this challenge, it became an opportunity as BugBot would now be anchored to another code repository site; BitBucket, and simultaneously be requested to work on GitHub repositories. This immediately expanded BugBot’s capabilities and market rather than limiting them. GitHub users can now opt-in to BugBot’s services by using one of two direct request methods. That is, the BugBotFINDME.txt file and the website direct request functionality.

5.1.2 Bug Gathering

Another challenge facing BugBot was its dependence on predefined bugs. Predefining all possibilities of bugs and further testing them to avoid false positives is a time-consuming process. The bugs currently in the BugHive were largely sourced from FindBugs open source bug collection, however, FindBugs only provided a description of the bugs and not the implementation in the form BugBot required so these bugs had to be effectively translated with respect to the criteria outlined in the BugHive.

In addition to sourcing bugs from FindBugs, I decided to develop a website with the goal of enlisting help from the coding community to accelerate the predefining process. This provides a submission form to users with all the criteria BugBot requires. So far, there have been two viable submissions.

Despite this, the BugHive is scarce with only 12 bugs, 8 for Java and 4 for C. This was due to the length of time it took to test each bug and ensure its correctness before being scanned for. In section 5.4.1 I discuss the future bug additions that can be added to the BugHive but that require more testing.

5.1.3 Creating a Presence

An overall success measure of BugBot would be its ability to create a presence in the wider programming community. Although this was not necessarily achieved during the time restrictions of this project, I have high hopes that BugBot will be a much used and well regarded static code analysis tool when all future additions, discussed in section 5.4, have been achieved.

As the website provides a great deal of content to users, background, design, and technical information, it has been speculated that posting the website URL to the right forums and threads online would generate interest and page visits.

This, in addition to BugBot's Twitter account, are efforts to create BugBot's presence. That being said, for BugBot to gain traction and be well regarded by programmers, it is vital that the BugHive be expanded to at least comprehensively cover one programming language.

5.2 ISSUES & LIMITATIONS

5.2.1 Bot Limitations

Bots and web-crawlers are generally talked about from a negative perspective, this could be due to the widely recognised term “spam-bots” or the worrying sense of the word “web-crawling”. For whatever reason, many web-users and more specifically developers have taken measures to ensure bots or web-crawlers do not access their information- webpages and code repositories alike, in the form of a robots.txt file. This file outlines the bots or web-crawlers disallowed entry to the directory.

To ensure BugBot was not classed as a malicious bot that developers would rather block than request, it was imperative to strike a balance between politeness and scalability, eagerness and download speed, and completeness of crawl and obeying robots.txt.

In addition to acquiring this balance, in order for a bot to make assumptions on human written code it must have a solid basis to make judgement and subsequently change code. The only way of ensuring this was by extensive testing of the bugs before being added to the BugHive and testing of the source code.

5.2.2 Limitations of Bug Identification

A crucial element to the success of BugBot is avoiding false positives. Many static code analysis tools are subject to the same issue, as FindBugs, as discussed in section 2.4 reports a false positive rate “below 50%”, not necessarily an ideal figure.

False positives are the identification and subsequent “fixing” of a bug that, in reality, was not a bug. BugBot to some extent depends on a user’s generally good programming practice to ensure “wanted cases” of bugs are not a reality. The obvious challenge here is that some bugs are features, and behaviour that constitutes a failure in one system may be a feature in another.

The compromise to this issue was adding “Wanted cases” of a bug as a criterion to the BugHive. A “Wanted Case” would be a logical reason for the bug being present. Such as the case of the casting

Math.random() bug, the only possible wanted case of this bug would be a desired integer output of 0. In this case it is safe to assume that if a programmer wanted a 0 integer; this is not how they would declare it. Realising this involved extensive brain-storming into any possible “wanted cases” of a bug and precautionary measures to ensure if a false positive or exception does arise; the Bot acknowledges responsibility and takes preventative measures for future fixes. This service was provided to users on the BugBot.org website through an “Issues” submission form.

5.3 FUTURE OF BUGBOT

Although BugBot has progressed greatly through the development of this project, it is far from complete. Below are few of the many ambitions BugBot strives to achieve to truly become a successful automated static analyser of online code repositories.

5.3.1 Future Bug Additions & Language Capabilities

Future bug additions and language capabilities are essential for BugBot to be a popular tool for users. Although bugs were hypothesised and considered, it eventually became evident that time restrictions would hinder the addition of these bugs. Below are a list of bugs and a general description of how they would be implemented in BugBot, but would require further source code abilities and testing to be validated and added to the BugHive.

1. JAVA, CASTING INTEGER MULTIPLICATION TO LONG

Overview: The multiplication of two large integers can result in an overflow (see Abstract for real occurrence of this bug). Java's integer values are 32-bit data types, with a minimum -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive). If multiplying two large integers and assigning them to a long variable, the long variable will be the overflow value rather than a value larger than the 32-bit value. Casting the multiplication can fix this.

General String: `long l = [int]*[int];` `// in a case where integers are large`

Fix: `long l = (long) [int]*[int]`

2. JAVA, WRONG EQUALS METHOD USED ON ARRAYS

Overview: Often times the “.equals()” method for comparing character sequences of strings is mistakenly used to compare contents of arrays. When used on arrays this method returns the same result as “array==array1”, comparing array pointers. To compare the array contents Java uses a method; `Array.equals(array, array1)`. BugBot would have to make the assumption that programmers intend to compare the contents of the array and not array pointers.

This bug could be truncated into many as other frequently used methods such as the toString() method. When toString is called on an array it would print the array pointer rather than the array contents. To implement these bugs, BugBot would require source code to identify variables of type Array.

General String: myarray.toString()

Fix: Array.toString(myarray)

3. JAVA, NOT CLOSING STREAMS

Overview: This bug presented itself as a featured question on StackOverflow [60], it seemed many beginner programmers were unaware of the need to close streams after opening. If appropriate additions were made to the BugBot source code to handle a different type of bug, a check could be implemented whereby each opened stream should be checked to ensure it was closed before the end of the program. To fix the bug BugBot would require enough information about the program flow and structure to know where the stream should be closed.

4. PYTHON & JAVASCRIPT LANGUAGE CAPABILITY

As discussed in section 3.4.2, BugBot would benefit greatly by adding language capabilities for both the Python and JavaScript programming languages. In the data compiled by GitHub, figure 30, it could be speculated that the number of repositories BugBot could scan would triple. Currently, from BugBot's cloned repositories 65% of these are declared unsupported languages in the metadata of the repository, however, to be noted in relation to this is that 10% of the repositories labelled unsupported did in fact contain files of a supported language.

When a sufficient number of Java and C bugs have been implemented in BugBot, the next step would be to implement JavaScript followed by Python.

5.3.2 Integration of FindBugs

As FindBugs operates under a Lesser General Public Licence and is a popular and proven Java static code analysis tools, BugBot would benefit greatly from implementing the bugs FindBugs defines in its source code through regular expression definitions in the BugHive.

FindBugs scans for a total of 424 Java bugs, although some are much less critical than others i.e. a method name beginning with an upper-case letter is defined as a bug, much of the critical bugs would be incredibly valuable to BugBot's capability and expansion. However, the major differences between FindBugs and BugBot in its bug identification process is that BugBot implements its bugs dynamically through a database and aims to keep the source code relatively short and handle each bug similarly, in contrast, FindBugs defines each bug separately through its source code. Running Source Monitor on the FindBugs source code; results showed the project to be over 333,000 lines long dispersed among nearly 3000 classes.

Another major difference between BugBot and FindBugs that would be critical to its integration is that BugBot aims to automatically fix the bugs it identifies – within reason. In order to do this, BugBot requires extensive testing to ensure the bugs it is fixing are in fact bugs and additionally ensure the bug is fixed correctly. The integration of FindBugs would therefore require each of the critical bugs to have a corresponding “fix” by BugBot's definition standards. This, as mentioned previously is the reason BugBot's expansion of the BugHive is relatively slow.

If BugBot were to adapt and implement FindBugs it would only include the critical bugs that are considered semantic errors and return unwanted or unexpected results rather than preferable naming conventions and constructs.

5.3.3 Cloud Based Deployment



For continuous deployment of the bot a local device will not suffice, but instead use some form of cloud service platform. Microsoft Azure, Amazon Web Services and Heroku are all forms of cloud service platforms that offer very similar functionality to deploy code.

Microsoft Azure allows deployment of a bot from Microsoft's Visual Studio as an end-point. With this functionality, it allows a developer to send requests and receive responses to an endpoint on a local-host. Azure can also run java programs, monitor a website and store an SQL database.

Amazon Web services [61] allows for fully automated code deployment through its AWS CodeDeploy service. This allows you to deploy from one to thousands of instances depending on the desired scale. It also facilitates rolling updates across all instances and code deployments can be stopped or rolled-back if any errors exist.

Heroku [62] is a cloud platform based on a managed container system. Heroku Run-time allows developers to deploy code continuously without any manual intervention. It offers a free limited sandbox ideal for experimenting during the initial testing of the Bot deployment.

Further investigation and testing at a later stage of the project development will result in a choice of which Cloud based deployment service is best suited to this project.

5.4 ACHIEVEMENTS

BugBot is a hugely rewarding project that has proved successful in its aim to save developers time by automating the debugging process and improving their code with no effort required. By identifying and fixing bugs across 11 repositories from the 1000 repositories scanned, it has proved to be a useful and capable tool that is needed by developers using source code management websites.

Despite BugBot's limited collection of bugs to scan for it identified 42 bugs in the 100 repositories scanned. If the BugHive was expanded to firstly cover the existing languages; Java, C and C++ comprehensively and further include python and JavaScript capabilities the bug finding results would increase exponentially.

BugBot is far from complete but has provided proof that an automated static code analyser is a viable and required resource among remote repository users and in the six-month time-span of this project, I am satisfied with the foundation that has been built for the project's future development.

As of the 29th March 2017 the following figures summarise BugBot's progress in 6 months of development and less than a month of scanning with all key features.

BugBot's Progress	
Bugs in BugHive	12
Languages Supported	Java, C, C++
Repositories Scanned	1008
Files Scanned	12069
Variables Identified	13,552,844
Lines of Code Scanned	2,948,415
Total Time Scanning	18hours 21min
Bugs Found	42
Fixed Repositories	11

Table 8 BugBot's Progress

REFERENCES

1. Jazequel, J. and Meyer, B. (1997). Design by contract: the lessons of Ariane. *Computer*, 30(1), pp.129-130.
2. Hailpern, B. and Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, 41(1), pp.4-12.
3. Research Triangle Institute. (2002). *The economic impacts of inadequate infrastructure for software testing*. National Institute of Standards and Technology.
4. Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. (2005). *Perspectives on free and open source software*. 1st ed.
5. *2013 Open Source Report*. (2013). Coverity Scan by Synopsis.
6. Wagner, S., Jürjens, J., Koller, C. and Trischberger, P. (2005). Comparing Bug Finding Tools with Reviews and Tests. *Lecture Notes in Computer Science*, pp.40-55.
7. Findbugs.sourceforge.net. (2015). *FindBugs FAQ. (Q2)* [online] Available at: <http://findbugs.sourceforge.net/FAQ.html> (Accessed 23 Mar. 2017).
8. GitHub. *Build software better, together*. Available at: <https://github.com/> (Accessed: 1 November 2016)
9. BitBucket, Atlassian. *The git solution for professional teams*. Available at: <https://bitbucket.org/> (Accessed: 1 November 2016).
10. MySQL (2016) *The world's most popular open source database*. Available at: <https://www.mysql.com/> (Accessed: 1 November 2016).
11. Bugbot.org. (2017). *BugHive – BugBot*. Available at: <http://BugBot.org/BugHive> (Accessed 22 Mar. 2017).

12. Twitter.com. (2017). *Sign up*. Available at: <http://Twitter.com> (Accessed 22 Mar. 2017).
13. Bugbot.org. (2017). *BugBot – The Bot that fixes Bugs in code*. Available at: <http://BugBot.org> (Accessed 22 Mar. 2017).
14. Levine, Sheen S. and Prietula, Michael, (2013) *Open Collaboration for Innovation: Principles and Performance*. Organization Science, Forthcoming.
15. Bitkeeper.com. (2017). *BitKeeper*. Available at: <https://www.bitkeeper.com/> (Accessed 30 Mar. 2017).
16. Kernel.org. (2017). *The Linux Kernel Archives*. Available at: <https://www.kernel.org/> (Accessed 22 Mar. 2017).
17. Hinsien, K., Läufer, K. and Thiruvathukal, G. (2009). Essential Tools: Version Control Systems. *Computing in Science & Engineering*, 11(6), pp.84-91.
18. Subversion.apache.org. (2017). *Apache Subversion*. Available at: <https://subversion.apache.org/> (Accessed 22 Mar. 2017).
19. Collins-Sussman, B., W.FitzPatrick, B. and Pilato, C. (2017). *Chapter 9. Subversion Complete Reference*. [online] Svnbook.red-bean.com. Available at: <http://svnbook.red-bean.com/en/1.7/svn.ref.html> (Accessed 30 Mar. 2017).
20. Git-scm.com. (2017). *Git*. Available at: <https://git-scm.com/> (Accessed 22 Mar. 2017).
21. Mercurial-scm.org. (2017). *Mercurial SCM*. Available at: <https://www.mercurial-scm.org/> (Accessed 24 Mar. 2017).
22. Google Trends. (2017). *Google Trends*. Available at: <https://trends.google.com/trends/explore?date=all&q=git,mercurial,subversion> (Accessed 24 Mar. 2017).

23. Atlassian. (2017). *Git Workflows and Tutorials / Atlassian Git Tutorial*. [online] Available at: <https://www.atlassian.com/git/tutorials/comparing-workflows> (Accessed 24 Mar. 2017).
24. GitHub. (2017). *Build software better, together*. [online] Available at: <https://github.com/about> [Accessed 30 Mar. 2017].
25. Louridas, P. (2006). Static code analysis. *IEEE Software*, 23(4), pp.58-61.
26. IEEE. IEEE standard 610.12-1990, 1990.
27. Jackson, W. (2009). Static vs Dynamic Code Analysis. *GCN*.
28. Binkley, D. (2007). Source Code Analysis: A Road Map. *Future of Software Engineering (FOSE '07)*.
29. Hristova, M., Misra, A., Rutter, M. and Mercuri, R. (2003). Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin*, 35(1), p.153.
30. NetBeans © (2016) *Connecting to oracle database - NetBeans IDE*. Available at: <https://netbeans.org/kb/docs/ide/oracle-db.html> (Accessed: 1 November 2016).
31. FindBugs™ (2005) *Find bugs in java programs*. Available at: <http://findbugs.sourceforge.net/> (Accessed: 1 November 2016).
32. Vestola, M. (2012). *Evaluating and enhancing FindBugs to detect bugs from mature software: Case study in Valuatium*. Masters Degree programme of Computer Science and Engineering. Aalto University School of Science.
33. Kim, S. and Ernest, M. (2007). Which Warnings Should I Fix First? *In Proceedings of the Proceedings of the 6th joint meeting of the European software engineering conference*.
34. Deissenboeck, F., Juergens, E., Hummel, B., Wagner, S., Parareda, B. and Pizka, M. (2008). Tool Support for Continuous Quality Control. *IEEE Software*, 25(5).

35. Synopsys, Coverity. (2016) *Software testing and static analysis tools* / Synopsys. Available at: <https://www.coverity.com/> (Accessed: 1 November 2016).
36. Klocwork.com. (2017). *Source Code Analysis Tools for Security & Reliability* / Klocwork. Available at: <http://www.klocwork.com/> (Accessed 22 Mar. 2017).
37. ?
38. N. Ayewah, W. Pugh, J. Morgenthaler, J. Penix, and Y. Zhou. (June 2007) *Evaluating static analysis defect warnings on production software*. In Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering,
39. Scan.coverity.com. (2017). *Coverity Scan - Static Analysis*. Available at: <https://scan.coverity.com/> (Accessed 25 Mar. 2017).
40. Jlint.sourceforge.net. (2017). *Jlint - Find Bugs in Java Programs*. [online] Available at: <http://jlint.sourceforge.net/> (Accessed 22 Mar. 2017).
41. Dolan-Gavitt, B., Hulin, P., Kirda, E., Leek, T., Mambretti, A., Robertson, W. and Ulrich, F. (2016) *Large-Scale Automated Vulnerability Addition*.
42. Porter, B., Zyl, J. and Lamy, O. (2017). *Maven – Welcome to Apache Maven*. Maven.apache.org. Available at: <https://maven.apache.org/> (Accessed 22 Mar. 2017).
43. MacNeill, C. and Bodewig, S. (2017). *Apache Ant - Welcome*. Ant.apache.org. Available at: <http://ant.apache.org/> (Accessed 22 Mar. 2017).
44. Sourceforge.org. (2017). *SourceForge - Download, Develop and Publish Free Open Source Software*. Available at: <http://sourceforge.org> (Accessed 22 Mar. 2017).
45. Cloudforge.com. (2017). *Free Subversion and Git Hosting / Bug and Issue tracking / CloudForge*. Available at: <http://www.cloudforge.com/> (Accessed 22 Mar. 2017).
46. Lundblad, N. (2007). *e-Exclusion and Bot Rights*. 1st ed. St Anna Research Institute.

47. Visual Studio. (2017). *Visual Studio | Developer Tools and Services | Microsoft IDE*. Available at: <https://www.visualstudio.com/> (Accessed 22 Mar. 2017).
48. Microsoft (2016) *Microsoft azure: Cloud computing platform & services*. Available at: <https://azure.microsoft.com/en-us/> (Accessed: 1 November 2016).
49. GoDaddy. 2017. Domain Names | The World's Largest Domain Name Registrar - GoDaddy IE. Available at: <https://ie.godaddy.com/>. (Accessed 22 March 2017).
50. WordPress (2016) *Create a website or blog*. Available at: <https://wordpress.com/> (Accessed: 1 November 2016).
51. Campwoodsw.com. (2017). *SourceMonitor V3.5*. Available at: <http://www.campwoodsw.com/sourcemonitor.html> (Accessed 22 Mar. 2017).
52. Watson, A. (1996). *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. National Institute of Standards and Technology Special Publication 500-235.
53. McCabe, T. (1976). A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4), pp.308-320.16.
54. Diakopoulos, Nick and Stephen Cass. (26 July 2016.). *"The Top Programming Languages 2016"*. IEEE Spectrum: Technology, Engineering, and Science News.
55. GitHub Octoverse 2016. (2017). *GitHub State of the Octoverse: 2016*. [online] Available at: <https://octoverse.github.com/> (Accessed 22 Mar. 2017).
56. Eclipse. (2017). Available at: <https://eclipse.org/> [Accessed 30 Mar. 2017].
57. Docs.oracle.com. (2015). *Lesson: Regular Expressions (The Java™ Tutorials > Essential Classes)*. [online] Available at: <https://docs.oracle.com/javase/tutorial/essential/regex/> (Accessed 16 Mar. 2017).

58. Docs.oracle.com. (2017). *Double (Java Platform SE 7)*. Available at:
<https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html> (Accessed 30 Mar. 2017).
59. McMillan, R. (2012). GitHub Says ‘No Thanks’ to Bots — Even if They’re Nice. *Wired*.
Available at: <https://www.wired.com/2012/12/github-bots/> (Accessed 30 Mar. 2017).
60. Stackoverflow.com. (2017). *Stack Overflow*. Available at: <http://www.stackoverflow.com>
(Accessed 28 Mar. 2017).
61. Amazon (2016) *Amazon web services (AWS) - cloud computing services*. Available at:
<https://aws.amazon.com/> (Accessed: 1 November 2016).
62. Heroku (2016) *Cloud application platform*. Available at: <https://www.heroku.com/>
(Accessed: 1 November 2016).

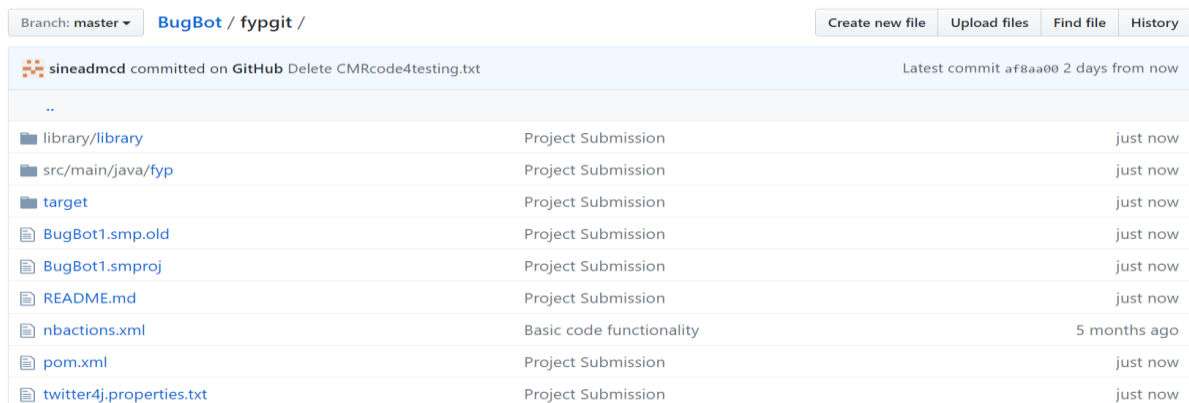
APPENDICES

All code for this project is open source at www.github.com/sineadmcd/BugBot.

The code here includes the Java code for the back end development of BugBot and the visual basic code for the administrators GUI.

The Java code is the commit labelled “Project Submission” and its root folder is fypgit. Within this folder is the further directory: src/main/java/fyp which contains all 11 source code classes as discussed in section 3.3.

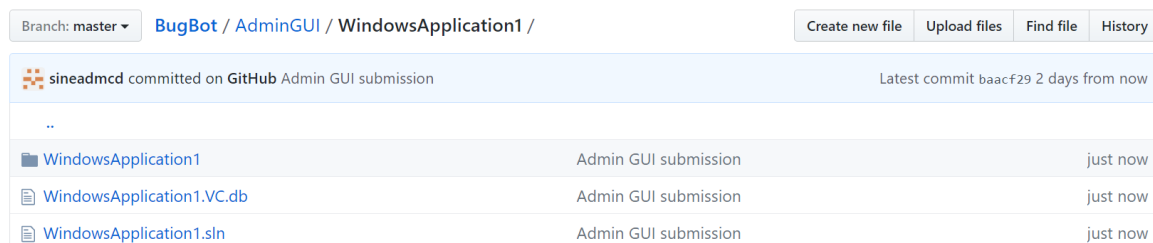
The SourceCode Monitor files are also contained in the fypgit directory, namely BugBot1.smproj.



Branch: master BugBot / fypgit /		Create new file Upload files Find file History
sineadmcd committed on GitHub Delete CMRcode4testing.txt Latest commit af8aa00 2 days from now		
..		
library/library	Project Submission	just now
src/main/java/fyp	Project Submission	just now
target	Project Submission	just now
BugBot1.smp.old	Project Submission	just now
BugBot1.smproj	Project Submission	just now
README.md	Project Submission	just now
nbactions.xml	Basic code functionality	5 months ago
pom.xml	Project Submission	just now
twitter4j.properties.txt	Project Submission	just now

Figure 67 Project Submission on *GitHub.com/sineadmcd/BugBot*

The Administrator GUI code and corresponding design files are in the folder labelled AdminGUI. All source code for each of the three different panels of the GUI are labelled as Form1.vb ; main panel, Form3.vb ; BugHive panel, and Form5.vb; Graphs panel.



Branch: master BugBot / AdminGUI / WindowsApplication1 /		Create new file Upload files Find file History
sineadmcd committed on GitHub Admin GUI submission Latest commit baacf29 2 days from now		
..		
WindowsApplication1	Admin GUI submission	just now
WindowsApplication1.VC.db	Admin GUI submission	just now
WindowsApplication1.sln	Admin GUI submission	just now

Figure 68 AdminGUI Submission on *GitHub/sineadmcd/GitHub*