UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

OLUWATOBI SAMUEL OMISAKIN

Relationship between Module Size, Alternative Cost and Bugs

Master's Thesis (30 ECTS)

Supervisor(s): Siim Karus

Tartu 2018

Relationship between Module Size, Alternative Cost and Bugs

Abstract:

The aim of this thesis is to find out if Alternative Cost (AC) and size of modules lead to more bugs in a software project. Using the historical churn extracted from revisions data and bug reports data retrieved from four software projects namely, JQuery, Font-Awesome, ReactJS, and Atom, we calculate their AC. After which we use Kendall correlation to investigate the strength of association between AC and bugs, and module size (measured in Lines of Code) and bugs. We find a strong association between size of modules in all four software projects and bugs existing in them, while that of AC and bugs remain inconclusive. From our investigation, we conclude that when quality assurance activities are performed on a software project, modules with larger size should be given more attention. On the other hand, using our result, Alternative Cost is not relevant for bugs localization.

Keywords:

Alternative Cost, Software Modules, Bugs, Software estimation, Impact analysis, Size-defect relationship

CERCS:

P170 Computer Science, numerical analysis, systems, control

Sõltuvus mooduli suuruse, alternatiivkulu ja vigade vahel

Lühikokkuvõte:

Selle lõputöö eesmärgiks on uurida, kas alternatiivkulu (AC) ja mooduli suurus viivad suurema vigade arvuni tarkvaraprojektis. Kasutades nelja tarkvaraprojekti – JQuery, Font-Awesome, ReactJS ja Atom – versiooniajaloost ja vearaportitest eraldatud andmeid, arvutame me nende alternatiivkulud. Seejärel kasutame me Kendalli korrelatsiooni, et uurida AC ja vigade ning mooduli suuruse (mõõdetuna koodiridades) ja vigade vahelise seose tugevust. Me leidsime, et moodulite suuruse ja vigade vahel on tugev korrelatsioon kõigis neljas tarkvaraprojektis. Samas AC ja vigade vaheline seos jäi tõendamata. Oma uurimusest järeldame, et tarkvaraprojekti kvaliteeditagamise tegevuste käigus tuleks suurtele moodulitele pöörata rohkem tähelepanu. Alternatiivkulu ei ole oluline vigade asukoha tuvastamiseks.

CERCS: P170

Võtmesõnad: Alternatiivkulu, tarkvara moodulid, tarkvara hindamine, mõjuanalüüs, suuruse-vea suhe.

Table of Contents

1 Introduction 8
2 Related works 10
2.1 General Bugs Localization Methods 10
2.2 Historical Bugs Localization or Prediction Methods 12
2.3 Alternative Cost and Modules 13
3 The method of result finding 16
4 The Projects Used 19
5 Data Collection 22
5.1 Bugs Data 22
5.2 Tools 23
5.2.1 GIT 23
5.2.2 GitHub 23
5.2.3 Laravel 24
5.3 Methods of Data Management and Retrieval 24
5.3.1 GitHub's API 24
5.3.2 Storage in MySQL database 26
6 Result of Analysis 27
6.1 Alternative Cost and Bugs frequency 27
6.2 Models for Estimation Calculation 28
6.3 Calculating the Module size of each project 33
6.4 Correlations 33
6.4.1 Correlations of Module Size and Bugs 35
6.4.2 Correlations of Alternative Cost and Bug fixes 39
7 Threats to Validity 43
8 Challenges 45
9 Conclusions and Contributions 47
10 Acknowledgement 48
11 References 49
Appendix 51
I. VCS_Modules Table Structure 51
II. VCSEstimations Table Structure 52
III. Project Resource link 53
IV. Script to calculate Threshold 54
V. Website Development History 57

VI. License 58

LIST OF TABLES

Table 4.1 The projects considered 19
Table 4.2 Summary of the projects used 20
Table 5.1Parameters for retrieving Issues from GitHub API 25
Table 6.1 Alternative Cost and Bug frequency 27
Table 6.2 List of variables used for estimations 29
Table 6.3 Alternative Cost Estimation Model Errors for all Metrics at Various Confidence Levels. 34

LIST OF FIGURES

Figure 2.1The method of fault localization 11
Figure 3.1 The flow chat of the process from start to end 17
Figure 6.1 Font-Awesome's level 2 Alternative Cost per module 28
Figure 6.2 JQuery's prediction model variables used. 31
Figure 6.3 JQuery's model Dependency Network at the closest node 32
Figure 6.4 JQuery's Module Size and Bugs correlation 35
Figure 6.5 Font-Awesome's Module size and Bugs correlation 36
Figure 6.6 ReactJS' Module size and Bugs correlation 37
Figure 6.7 Atom's Module size and Bugs correlation 37
Figure 6.8 The correlation between Module (Non-Compared) Size and Bugs 38
Figure 6.9 JQuery's AC and Bug fixes correlation 39
Figure 6.10 Font-Awesome's AC and Bug fixes correlation 40
Figure 6.11 React's AC and Bug fixes correlation 41
Figure 6.12 Atom's AC and Bug fixes correlation 42

1 Introduction

The increasing complexity of software systems has led to the increase of the need to address challenges related to software quality, because there are defects one way or the other which leads to failure in the concerned software (Koru, Zhang, Emam, & Liu., 2009). Researchers however, in a bid to understand and predict bugs in software have attempted to identify software properties that correlate with fault-prone modules for many years (Bell, Ostrand, & Weyuker, 2006). They have come up with methods and techniques using different models, such as Negative Binomial Regression (Ostrand, Weyuker, & Bell, 2004), Relative Code Churn (Nachiappan & Ball, 2005) to help concerned stakeholders, developers and testers understand where to put their effort in the process of bug fixing or software testing. Most of them makes use of module size measured in number of Source files or Source Line of Code (SLOC) as common metrics for finding fault prone parts of a given system. The result of their finding has produced mixed evidences of the correlation of the module size and bugs, that is some argue that exponential increase in bugs does not come from the increase in module size while others oppose this.

(Karus, 2014) explored the use of Alternative Cost (AC) in finding generated parts of different open source software. The discussions and finding highlights the fact that negative AC shows the possibility of a module being generated. Based on their work, we follow the same usage of Alternative Costs, that is, the costs of replacing a code module (Karus, 2014). We aim to apply their method of quantified estimation to find out the relationship module size has on the number of bugs in a software, and the relationship the AC of a project's modules has with bugs in the software. Therefore, our research question is broken into two:

1. What effect does module size have on the size (extent or number) of bugs (or defects) in a software system?
2. What effect does Alternative-Cost (AC) have on the size of bugs in a software system?

The hypothesis we propose is that Alternative Cost and module size tends to lead to more bugs in a software project.

Four (4) open source projects are used for this work: JQuery, Font-Awesome, ReactJS, and Atom – all of which are web-development related projects and are hosted on GitHub. Atom is a text editor for programmers; ReactJS is a Javascript library for developing reactive web application; Font-Awesome is a front-end library that provides Icon for use on web pages; while, JQuery is a JavaScript library that helps to work with DOM manipulation.

Web development related projects were selected in order to keep the type of projects as close as possible and also because the web-development cuts across modern development practices, that is, with JavaScript, CSS, and HTML now runs on their own "OS" called Web browser (Peter, 2013). Furthermore, web-development has become easy that mobile and desktop app can be developed starting from using a Text Editor (Mozilla Firefox, 2018).

The advantage using open source project on the other hand is because open source projects aid research improvements since this does not prevent researchers from repeating or extending the research (Rosenberg, 1997).

This work comprises of five major steps:

1. The review of literature: we present a review of literatures of bug detection mechanisms or techniques. We consider understanding what Alternative Cost is in Software System. Then, we review relevant literatures about module size relation with bugs (or defects).
2. The data Collection, manipulation, and preparation: The data needed for this work are: Bugs (or issue) reports data from their respective repositories; The historical data of revisions from Git Version Control System (Hosted on GitHub). Here we also prepare the data for analysis.
3. The Analysis: Using the Decision Tree Algorithm of Microsoft SQL Server Analysis Services[1] we find out the estimation based on the Yearly Lines of Code (LOC) Churn of the modules in the system. Code churn is a measure of the amount of code (LOC) change taking place within a software unit over time (Nachiappan & Ball, 2005).
4. The result calculation: We calculate the Alternative Costs derived from the estimations of each project's yearly churn, and their correlation with the bugs retrieved per module date. Also, we find out the size of modules at those given date instance, and use that to calculate their correlation with existing bugs.
5. The result presentation: We present the result of the correlation of the modules AC, bugs, and size across the testing dates of the software lifetime considered.

In the course of this work, the term "bugs", "defect", and "faults" are used interchangeably.

[1] https://docs.microsoft.com/en-us/sql/analysis-services/analysis-services?view=sql-server-2017

2 Related works

Bug, smell, or defect can be anything that doesn't conform to standard set for it. The quality and productivity of software are affected when there are bugs in software (Jones, Harrold, & Stasko, 2001). The quality of programs or software apart from fulfilling its business requirement is dependent also on the minimal number of bugs it consists. Furthermore, "software defects are introduced" (Wong, Gao, Li, Abreu, & Wotawa, 2009) in many ways when activities are performed on the development of the software. (Yasmeen, 2014) calls it error, flaw, mistake failure, or fault making the system or software to function abnormally (that is, away from the expected behavior). Bugs existence in a software is not dependent on the level of experience of a developer only, because due to the limitation of human even when experienced developers code the mistake they make could throw up errors in unexpected place during software testing (Roychoudhury, 2010).

The process of finding those faults in software could be very expensive depending on the size of the software project. Not only because of the money invested into tracking and its fixing, but even because of the time spent and the risk of breaking other part of the software in the process of fixing the found bug. In this section we consider Localization, that is, some of the common methods used apart from historical investigation to find location of bugs in a software. Next, we look at how historical data has been used to predict/detect defect-prone modules, and how important AC may be in fault localization.

2.1 General Bugs Localization Methods

Fault localization are methods by which faults, bugs, or defects are searched for in a software. It encompasses "the activity of identifying the exact locations of program faults" (Wong, Gao, Li, Abreu, & Wotawa, 2009). The "activity" in this definition shows that there are steps or processes that lead to locating those bugs. Localization furthermore helps find or make timely identification of fault prone modules (Yasmeen, 2014) so the likelihood of its escalation is prevented. Various activities including testing, debugging, issue tracking, exploration, and the likes are summed up under localization.

Debugging is one of those processes that is involved in bugs localization when there is a loss of data, breakdown of software functionality or its quality reduction. According to (Jones, Harrold, & Stasko, 2001) debugging is expensive. This is understandable because debugging method comes just after an abnormality has risen unlike methods that may be used before fault arises such as visualization, constant analysis of the condition of the given software, and data mining to observe historical pattern of behavior of the software to help

prevent or avert the circumstances that may come up. One of the methods adopted when debugging is software testing. It exposes areas that does not work based on requirement and it helps keep the software stable even though testing may be done before and after the release of the software.

There are other processes involved in fault localization mentioned by (Wong, Gao, Li, Abreu, & Wotawa, 2009) which includes data mining techniques, visualization (example with Tarantula[2]), and Model-based type which establishes the relationship between the outcome of a test and its types using models. (Roychoudhury, 2010) gave an insight into using Trace comparison, for example in fault localization. The process explains the efficiency of trace visualization which they confirmed as not enough in localization of bug. The principle behind this technique results to the investigation of bugs by considering passed tests against failed tests (i.e. the traces). Their work highlights the weakness of localization of bugs when comparing successful trace or tests execution against the failed ones.

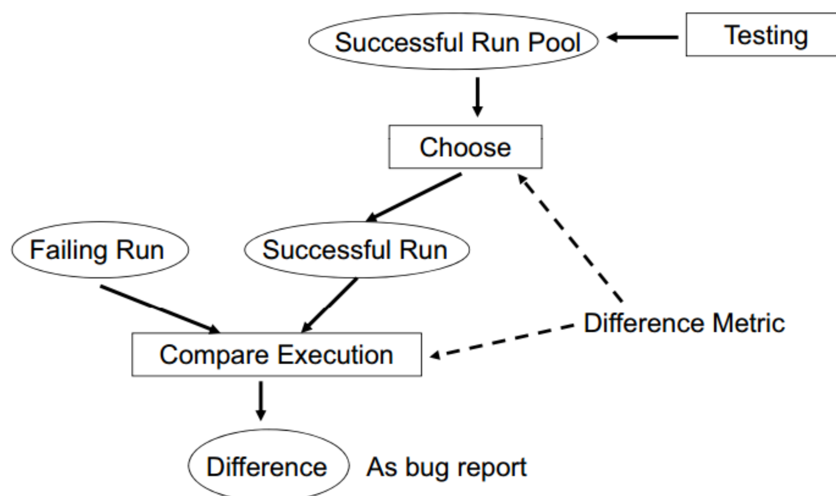


Figure 2.1The method of fault localization

Source: (Roychoudhury, 2010)

Furthermore, the above (in Figure 2.1) an example given by (Roychoudhury, 2010) provides pointers that helps make assertion that a software modules or functionality works as expected using sets of test cases. Given a set of test cases that is expected to conform to a result, when some of the tests fail, if they are in similar range or boundary values with other passed tests then the failed test can reveal where in the code the bugs are. The choice of

[2] https://www.cc.gatech.edu/~stasko/papers/icse02.pdf

passed test could be difficult, however if their boundary values are considered then it can be made easy. The comparison of the execution of the tests are then logged as a Bug/Issue report.

2.2 Historical Bugs Localization or Prediction Methods

The composition of a module is the files it contains, and the composition of a software project (source) file is its lines of code. There are number of studies that have investigated the relation between the modules and defect (or bugs). One which is like ours is (Malaiya & Denton, 2000) where they presented a model to show how the size of modules affect the density of defect in several projects with the aim of reducing the number of defects. They looked at the density of defects in modules of varying sizes and further considered the possibility of minimizing the defect density of the modules (Malaiya & Denton, 2000). They derived a model that presents the observable characteristics of the relation between the defect density due to model size variation. Observing the modules' exponential distribution and probing its applicability, they concluded that a reduction in defect density may be achievable if "small modules can be combined into optimal sized modules" (Malaiya & Denton, 2000).

On the other hand, (Koru, Zhang, Emam, & Liu., 2009) in their Investigation into the Functional Form of the Size-Defect Relationship for Software Modules found out that there is correlation between bugs and module sizes using classes and defect data. They applied Cox model to four large open source projects namely, Mozilla, Cn3d, JBoss, and Eclipse which are Concurrent Versions System (CVS)[3] based. Similarly, to our work they used the revision history of the Version Control System of each project to find out bugs related commits. They found out that smaller modules tend to contain more bugs than larger modules. The conclusion is that given a quality assurance process with limited resources, their work would help to expend these resources in the right way by considering smaller modules first.

Contrary to our work however, they considered commits messages in the version history of the projects by finding keywords such as 'bugs', 'defects', and 'bug' to know when a bug is fixed (or when it is found). We classify bugs as those that are reported by a Pull Request (PR)[4] or Change Request (CR) in the Git[5] repository and those that were merged and closed.

---

[3] https://www.nongnu.org/cvs/
[4] https://help.github.com/articles/about-pull-requests/
[5] https://git-scm.com/

Although we do not only consider those with bug label but also took PR's that has 'Fix' in their titles. Their method has the possibility of introducing a wrong number of defects since there is a likelyhood that such commit message could:

1. Be made based on a near immediate mistake made by the developer especially with using the word 'fix' in a commit.
2. have repeated commit messages.

This drawback is handled by our study where we do not consider commit-level bugs but those based on the PR (which results to one or more commits made to the project). This approach ensures that, only those that are truly found to be bugs are used even though the corresponding commits which addresses the PR doesn't have for example 'fix' as part of their names. This is explained further in Section **Error! Reference source not found.**.

In the study modules relation with bugs in them, (Koru & Emam, 2009)'s work about "The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules" built on their previous work (Koru, Zhang, Emam, & Liu., 2009) that is to further prove that in contrast to the general idea that module sizes and complexity of modules are proportional to the number of defects, that smaller modules are to be given more priority when activities involving quality assurance is performed. They did this by observing where code coupling is higher by considering how refactoring actions affect the size-coupling in the projects and using their result to find out classes with code smells. Furthermore, by adopting Concentration curves they exposed three metrics: Coupling Between Object classes (CBO)[6] and Depth of Inheritance (DIT)[7] and LOC they conclude that smaller modules are more coupled and deduced that those modules are "proportionally more defect-prone".

2.3 Alternative Cost and Modules

Alternative Cost (or Opportunity Cost) generally according to (Buchanan, 1991) is the evaluation placed on the most highly valued alternatives or opportunities. It is the answer to the question, "what is the value of the alternative of the choice made?". The scarcity in resources ensures there's a need to make choice, comparing more than one decision we can tell by the observation of one of the available options if a person has made the right choice instead of the other. The consequence of the choice we made does not necessary show the

[6] https://maisqual.squoring.com/wiki/index.php/Coupling_Between_Objects
[7] https://blogs.msdn.microsoft.com/zainnab/2011/05/19/code-metrics-depth-of-inheritance-dit/

evaluation of the cost/value the other choice would have given rather the value observed at the time of decision is what the alternative cost is.

The importance of alternative cost to a software project aids the evaluation of the worth of individual parts in the project. For example, for this work we investigate if it would cost less to replace a module or it would cost more. It may also be an indicator of why a module was preserved or removed over the course of the duration of development, or otherwise why it has had few modifications. It also helps to understand what may likely befall those modules in the future especially using the historical distribution of the cost. This observation helps us consider what impact the size of the module is in relation to its alternative cost.

One of the several are various metrics which may be used to calculate the AC of a software project is LOC. Although it's been argued how the use of Lines of code may not be a viable measure of the efforts on a project (Yinhuan, Beizhan, & Yilong, 2009) however in recent study (Karus, 2014) the cost of replacing a project's module was modelled to help understand the quality of each module and this cost was derived from Lines of code.

We employ in this work the same technique used in (Karus, 2014) for estimating the cost of modules of the 4 projects used. The steps taken to achieve this cost estimation is detailed in section 9.

Apart from understanding how the alternative cost and module size of a project explains the valuable parts of a software project, we also consider how these cost and size may help understand the rate of bugs in the system. In their paper (Ostrand, Weyuker, & Bell, 2004) talks about localization of bugs in a software system by using the "faults identified at all stages of the development starting from the requirement phase" and at other different stages. Their method proposes the use of negative binomial regression, most importantly to predict the location of bugs in subsequent releases of a software project obtained from the fault data history gathered on files. The purpose of their work is quite similar ours that is, to use their technique to ensure developers and "testers can focus their effort" where important, and we aim also to help developers (testers inclusive) use the result of this work as a pointer to understand how the size of the module (and/or Alternative Cost) may be an indicator for the presence of bugs. However, this work does not entirely aim to provide a complete alternative to (Ostrand, Weyuker, & Bell, 2004), in fact it seeks to improve upon it in a way by using size of estimated modules to prove if they correlate to the module size or not.

Another study which adopts the metric of "code churn to predict the defect density early in a software system" (Nachiappan & Ball, 2005) stressed the advantage of using relative code churn against absolute code churn in the prediction of bugs using the releases of a software

system. Furthermore, just like (Ostrand, Weyuker, & Bell, 2004) they attempted to provide solution to future localization of faults (or bugs). Also, by calculating the correlation between actual and estimated defects, their prediction technique affirmed a relation between both variables, they distinguished "between fault-prone and none fault-prone binaries" (Nachiappan & Ball, 2005). Although our aim is similar with respect to bug because we also used the code churn to estimate our alternative cost by observing yearly activities on a project and which is a useful tool to also predict fault-prone modules based on its correlation with the characteristics of the modules. However, ours is focused on evidence we can derived from the correlations of Alternative Costs and size of modules with the number of bugs, which consequently prove the influence both Alternative cost and the size of the modules have on bugs present and the number of bugs that may arise in the system.

3 The method of result finding

We consider four open source projects and fetched their revision history, we find bugs reports from the project's respective issue tracking system using their fix we then are able mark out the modules that were fixed in the process. We then calculate the correlation between the modules (size and AC) in the projects and the bugs frequency using Kendall rank correlation coefficient[8].

Kendall correlation is chosen because of many reasons which includes the type of our data. Pearson correlation[9] doesn't fit well with non-uniform data, it is sensitive to outliers, and doesn't help test non-linear relationship between to data well. Spearman's correlation[10] would have come handy but has little difference to Pearson correlation apart from its ranking. Kendall is useful to test the dependence we have between the two non-related and non-monotonic variables (James, n.d.).

We consider GIT project in this work, and took advantage of GitHub's issue tracking system. Since GitHub provides a feature to manage issues related to each repository, the issues are mined by making API calls to GitHub Version 3 rest service. The steps undertaken is detailed in Section 5**.**

[8] https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient
[9] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
[10] https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient

Figure 3.1 The flow chat of the process from start to end

As shown in Figure 3.1, the process we underwent to perform the analysis of this work involves 9 steps. The organization of the data for example (step 3) involves various activities to get the historical data in their right format, and get the values of the Yearly churn based on the revision records retrieved from GitHub. We also at this stage built up the VCS_Modules and VCSEstimations table which is necessary for the prediction and analysis performed on the Churn values. The structure of this table is enlisted in Appendix I and II.

Comparing the issues fixes and modules checked

Using Kendall correlation, we find the relationship between the modules of these projects and their corresponding bug fixes at the dates of the alternative cost. Right before that, we make an estimation on the projects using the Yearly Lines of Code (YLOC) as the variable for the predictions. This enables us to calculate the Alternative Cost (AC) for each module that were present for our test data. The result, thus, helps us analyze the impact the value of the AC has on the possibility of having more bugs in these affected modules within a year period. In addition, Kendall's correlation is necessary to test the association's strength between the two variables.

Our conclusion thus, depends on the result of the correlation. We observe the *statistical significance* of having modules, alternative cost and bugs fixes correlated. After we perform the comparison of all modules present in the considered dates of the AC using relevant thresholds for significant difference. The result of the Kendall correlation is then plotted for conclusion.

4 The Projects Used

Some of the most popular open source projects were chosen. These have thousands of contributions from the communities of developers around the world. Here is the list of the projects with their repository URL:

Table 4.1 The projects considered

SN	Organization	Name	URL
1	JQuery	JQuery	https://github.com/jquery/jquery
2	Fort-Awesome	Font-Awesome	https://github.com/FortAwesome/Font-Awesome
3	Facebook	React	https://github.com/facebook/react
4	Atom	Atom	https://github.com/atom/atom

These projects have hundreds on contributions and are well maintained, with average age of 7 years, all of which are web-based or related projects. Further details about these projects are discussed in Section 0.

Projects Overview

The overview of the projects considered is gathered by GitStats[11], a statistics generator for git repositories (GitStats Repository, n.d.) – A tool that helps find some statistical details about a GIT project including the general activities.

JQuery

"jQuery is a fast, small, and feature-rich JavaScript library" (jQuery Homepage, n.d.). It is built to help with DOM (Documents Object Model) manipulation also provides method that enables the use of asynchronous interaction through ajax call to a server. It also exposes Application Programming Interface (API)'s that is extended on the core JavaScript. It has a simplicity of usage within its instant loading of the web document, and provides several functions to access underlying DOM objects and do even more. JQuery's main programming language is JavaScript.

Font-Awesome

Font-Awesome known as "Iconic Font and CSS (cascading stylesheet) toolkit" (Font-Awesome's website , n.d.) is icons library with hundreds of shapes. These icons are

[11] https://github.com/dmitryn/GitStats

generally used by designers of web applications to beautify their works. Font-awesome is managed on GitHub and communities of developers contribute to this open source project and the latest version as at the time of this work is 4.7.0.

ReactJS

"In 2013, Facebook released React" (Artemij, 2015). React, React.js, or ReactJS is "a JavaScript library for building user interfaces" (Homepage of React., n.d.). React emerged to make rendering the view of web pages based on a change that is made in the document. With the use of a method called "render", React can build up a component as simple as displaying a text in an HTML's DIV tag by taking "input data and returns what to display" (Homepage of React., n.d.). This makes use of the property of the main DOM object, and injects the new data.

Atom

"Atom is a desktop application built with HTML, JavaScript, CSS, and Node.js integration. It runs on Electron, a framework for building cross platform apps using web technologies." (Homepage of Atom, n.d.).

The quick overview of the project is shown in Table 4.2 Summary of the projects used.

Table 4.2 Summary of the projects used

Project	JQuery	Font-Awesome	ReactJS	Atom
From	22-03-2006	17-02-2012	2013-05-29	2011-08-19
To	12-01-2018	07-12-2017	2018-01-24	2018-01-18
Age in days	4315 (12 years approximately)	2121 (6 years approximately)	1701 (5 years approximately)	2345 (6.5 years approximately)
Active days	1851 (42.90%)	335 (15.79%)	1521 (89.42%)	2095 (89.34%)
Total Files	266	789	765	729
Total LOC	61066	1271	109817	464250
Added LOC	285745	161264	627982	1727198
Removed LOC	224679	159993	518165	1262948
Total Commits	6303 (average 3.4 commits per active day,	1161 (average 3.5 commits per active day,	9606 (average 6.3 commits per active day, 5.6 per all days)	34222 (average 16.3 commits per active day, 14.6 per all days)

	1.5 per all days)	0.5 per all days)		
Authors	309	96	1242	482

Its observed that as at January 2018 the average age of all the project in about 7 years and 4 months. The oldest of them is JQuery and the youngest is ReactJS. Using projects of varying years helps to determine how our work performs on projects of varying development duration and how the result of our investigation can be applied with less regard to the age of the project.

An important detail from the activities of the projects used is that the age of the project doesn't influence the files, Lines of Code, the number of commits. An example is that ReactJS is more active in terms of the commits, the number of files, and LOC it has within that short period of development than JQuery. ReactJS has the highest number of authors (or developers) amongst the four projects with 1242. The possible explanation to this is the fact that JavaScript development has become popular since ECMAScript[12] 5 was released (Peyrott, 2017) than when JQuery's development started, hence there are more JavaScript developers who are interested in contributing to modern libraries.

[12] https://en.wikipedia.org/wiki/ECMAScript

5 Data Collection

The data needed as previously stated are from GitHub. The two methods by which these projects are retrieved are:

1. Cloning the repositories
2. Server to Server call from the GitHub's API

As for the second method, the data retrieved through the calls to this API is persisted into a MySQL database, and the server side is implemented using Laravel's Lumen[13] PHP Framework.

The general process of collection involves two distinct result sets:

1. The repositories data such as commits, files, revisions, etc.
2. The issues linked with these repositories.

The further breakdown about how these are achieved is stated in sub-section 5.3 i.e taking advantage of GitHub's issue with repository structures.

5.1 Bugs Data

The general format or structures of the repository data includes, Commits, Pull Requests, Issues, Repository information such as creation dates, contributors, organizations etc.

There are some important factors considered when retrieving bugs data from the repositories. One of these is the need to distinguish between feature requests[14] and bug requests[15] itself.

One of the major indicator used to retrieve bug list from GitHub is the labels on Issue reports. GitHub has issues labels which are either strictly or loosely employed by various projects' maintainers. These labels largely help to understand how changes are made in the project and how collaboration is done. This consequently facilitates the solution to the challenges of knowing which reports are for bugs or feature.

In summary, we found out that labels including *bug*, *bug-fix, Bugs, Type: bug, and critical* are often used to identify bugs with the projects considered in this work. We further investigated how these labels can truly tell that these are bugs indeed, the evidence to strengthen these was that most of these issues have high closure without merging rate.

[13] https://lumen.laravel.com
[14] Feature request are those made to add new feature or functionalities to the Project.
[15] Bug request are the ones reported because of unexpected errors encountered when using the project.

After successfully understanding and listing these metrics for retrieving issues from repositories, we then filter out the issues which during the period considered triggers a pull request (PR). This means that some issues generally don't survive discussion phase possibly because they were later detected as arising because of the reporter's systems or setup faults. Finally, to understand what part of these filtered issues made it to the core source, we considered the Pull Requests that were merged to the main branch of the repository. Therefore, the true bug is the merged PR.

5.2 Tools

The tools that were employed in this project are the ones capable of doing three things in whole:

1. Request (from the API)
2. Storing (into files or database)
3. Retrieving (from database and process further)

As already stated tool considered such as Gitstats[11] is used in general to make surface evaluations of the project, such as counting lines of code and showing historical evolution of the code base.

5.2.1 GIT

"Developed in 2005, by Linus Torvalds", GIT version control system emerged to be a good and modern alternative to SVN, SubVersion, Mercurial etc. It is maintained as an open source project. The approach GIT provides is a workflow that takes changes and history as a pattern to manage the evolution of versions of projects. According to Atlassian, GIT is a Distributed Version Control System (DVCS) (Atlassian Tutorial - What is Git. , n.d.).

5.2.2 GitHub

GitHub hosts GIT projects and they provide easy User Interface (UI) and Application Programmer's Interface (API) to interact with projects stored on their website. One of the important features of GitHub that necessitates its choice in this work is because both the repository code base and the issue tracking system are maintained in easily accessible place. This means there is no need to navigate away entirely from the project's location when logging or addressing issues for that same project. This is a very important for easy management. Therefore, we could retrieve all the issues in the repository with their issues identifier easily mapped to a commit activity, and thus further ease the analysis constraints that could have been encountered if there would have been a need to jump to an external

issue tracking system. GitHub also exposes their API to make relations to issues and code base available in more flexible and understandable way.

5.2.3 Laravel

Laravel is the PHP framework for Web Artisans[16] (Laravel's homepage, n.d.). This framework is built on PHP language and intentionally made for building rich and fast web application. It is also coupled with feature-rich libraries and fluent management of dependencies. It has its own Active Record, management of migrations and models therefore helping to take care of situation where there is a need to resolve database interaction in batches.

Laravel is used in this work to serve as a more secure server-side platform for interacting with GitHub API, to host connection with the database system, and to also setup API needed to manage the entire process of communication with GitHub. Some API endpoints were written to dynamically manage the changes in project that is supplied. It is at these endpoints that such variable as project name, the labels, the limit of the data expected by GitHub is being passed, and where other needed parameters are setup before the final response is retrieved.

5.3 Methods of Data Management and Retrieval

The entire method adopted in the process of data retrieval are three, which are: Issue retrieval from GitHub’s API, Storage on MySQL database, Reforming of stored data. The following sub-section shows the details of these methods.

5.3.1 GitHub’s API

The GitHub’s API in version 3[17] exposes several endpoints to mine data from the repositories, both at the organizational level and individual user level. Meaning that repositories that are being managed by a known organization can be also interacted with through it. Even though the common interest of data miner would be to get data, the APIs provided by GitHub can also be used to almost make the CRUD[18] operations on their repository even though some actions are limited to authorized individuals. For this work, however we only need to retrieve data.

The API’s End-points

[16] Someone who does skilled work with their hands - https://dictionary.cambridge.org/dictionary/english/artisan

[17] https://developer.github.com/v3/

[18] Create Read Update and Delete (CRUD)

There are primarily two endpoints that are found to be interesting for getting the data needed for this work: The *search* and *repos* end points. While the *search* can be general, the *repos* end points are basically targeted toward specific repository, thus having higher rate limit than the search end points.

To be able to understand the work flow, we clearly state how issues are retrieved from GitHub and why they are retrieved is such manner. The basic parameters used for retrieving issues from GitHub API are shown in Table 5.1, using JQuery as an example.

Table 5.1Parameters for retrieving Issues from GitHub API

Parameter	**Value**	**Description**
state	closed	The state of the issues, either open or closed
sort	created	The pattern of sorting
direction	asc/desc	The direction of list from the oldest to the newest
since	2000-01-01T00:00:01Z	All issues that matches the parameters on and after the given date/datetime
per_page	100	Retrieve maximum (that is, GitHub's minimum) results
labels	Type: bug	Specifically, for JQuery's project, the labels are not prepended with *Type* (other projects may differ)
page	*2*	The next page in the paginated result

Using the API request parameters of JQuery, a sample API URL that helps retrieve the information of an Owner's repository: https://api.github.com/repos/jquery/jquery consisting of some important such as state (of the issues) and labels (of the issues). For retrieving issues, there are some peculiar properties of these projects especially in the way the issues are reported. One of these properties is the *labels*. JQuery's bug reports has the label *bug*. This property is peculiar to JQuery project; other projects also have their respective labels which helped in filtering their bug reports.

As previously introduced, this part presents the structure of how the API from GitHub gives access to the issues, the files affected by the commits on the concerned issue, and these are later processed to give the needed data used for the analysis this work presents. The brief overview of this structure is given in the following paragraph.

We first fetch the data about the repository concerned, then we search data within a timeframe so we can limit the data to the duration we want to consider. The importance of this is to be able to have a collective figure of duration this work focuses on which consequently would ensure that the factors considered in the final results are real, and so making the results weightier.

5.3.2 Storage in MySQL database

The data retrieved from API calls to GitHub is persisted into the database. During the process of saving these results there are also separations of data that are needed for the analysis. Generally, most popular database systems (apart from single file based such as SQLite) could have served the same purpose because the most important thing was to ensure that we can easily reuse these data without the need to continually depend on API calls to GitHub again. But the choice of this database tool built on InnoDB[19] engine, enforcing referential integrity, and having most command operations helpful and the fact that the developer is experienced more with PHP and MySQL.

5.3.3 Reforming the Stored data.

The data retrieved from GitHub are not in the format expected. There was need to reformat them. At a point, we organized the issue data to show the commits activities separating each data by the files that were affected and some calculations were done to determine their Yearly Lines of Code churn. All these were useful in calculating the estimations.

[19] https://dev.mysql.com/doc/refman/5.6/en/innodb-introduction.html

6 Result of Analysis

This section describes the result of the analysis done on the issues (bugs), alternative costs, and module size. The Kendall correlation between the alternative costs and issues are presented, and that of issues and module size that is, Lines of Code (LOC) of the module for each date.

For each date present in the alternative cost and for each module we find out the LOC for that module at the end of that day, and for that same date, we find issues count till 1 plus year; exceptional case in this situation is that all the counts are excluded for newer dates.

6.1 Alternative Cost and Bugs frequency

The module levels considered in the calculation of the correlation between cost and fixes are 2 to 4, the reason being that there are no changes in module level 1 that could help in comparison. Furthermore, correlations are not calculated for modules whose number of unique issues and values of unique alternative costs are constant. Finally, the AC data are ordered from oldest to latest by dates to see how the costs and issues evolved over the years. JQuery project has the max of 6 fixes or issues given the costs dates, meaning no date has more than 6 issues for their respective given modules. The minimum cost is -2091212 and the maximum cost is 0. The date span for the alternative cost considered in all levels 2, 3, and 4 is from 2013-01-08 to 2017-04-29.

Table 6.1 Alternative Cost and Bug frequency

Project	Max Bugs	Minimum cost	Maximum cost	Start date	End date
JQuery	6	-2091212	0	2013-01-08	2017-04-29
Font-Awesome	15	-5055156	13,098,032	2013-10-16	2017-03-15
ReactJS	216	0	6,421,757	2015-12-02	2017-03-28
Atom	50	-4287958	0	2016-07-20	2017-03-29

ReactJS project has the max 216 of fixes or issues given the costs dates. The minimum cost is 0 and the maximum cost is 6421757. The date span for the alternative cost considered in all levels 2, 3, and 4 is from 2015-12-02 to 2017-03-28.

Atom project has the max of 50 fixes or issues given the costs dates. The minimum cost is -4287958 and the maximum cost is 0. The date span for the alternative cost considered in all levels 2, 3, and 4 is from 2016-07-20 to 2017-03-29.

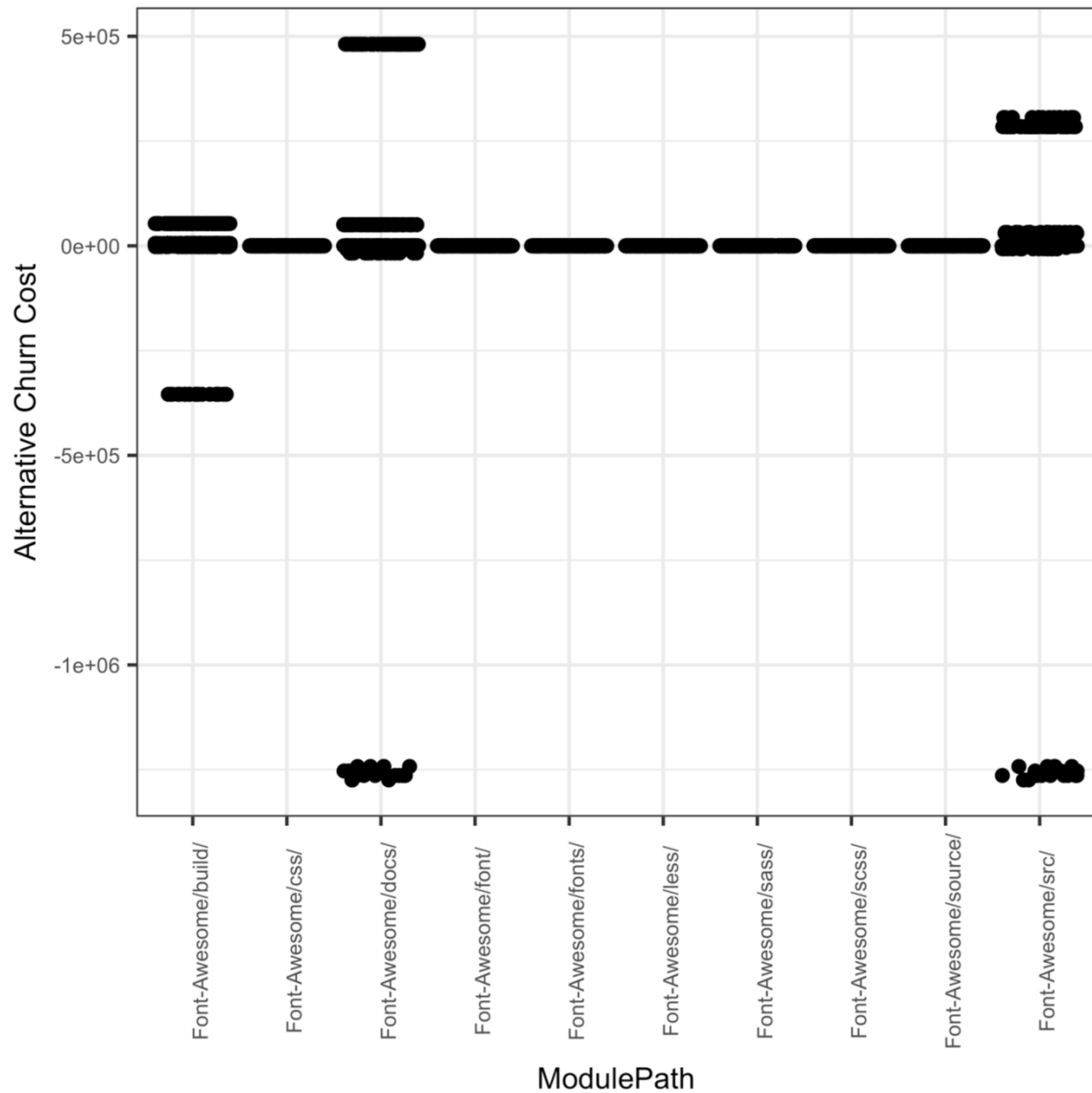


Figure 6.1 Font-Awesome's level 2 Alternative Cost per module

Font-Awesome project for example as shown in Figure 6.1 has the max of 15 of fixes or issues given the costs dates. The minimum cost is -5055156 and the maximum cost is 13098032. The date span for the alternative cost considered in all levels 2, 3, and 4 is from 2013-10-16 to 2017-03-15.

6.2 Models for Estimation Calculation

The process of calculating the alternative cost of each projects required an essential phase which is the calculation of the cost estimation. This phase is where the models used to define the estimation is built, and the result of the application of these models gives us the right estimation values to use in making the predictions, hence the alternative costs.

The models of each project are based on separating the revision history data into training and testing data. The training data constitutes the larger size of the project history: this is the 70 percent of the whole project's lifespan (that is, 70 percent of the revision history); on the other hand, the testing data comprises of 30 % of the project. Furthermore, the training data is the earliest dates of the development stage set at 70%, while the test data has newest dates. All project's repository revision data are retrieved up to 31st of March 2017.

This training data are mapped up to build the project's model, while the test data is used to evaluate the model's performance to make our prediction. The tool used is the Microsoft SQL Server (Analysis Services). This tool has various packages that allow importing database records, running analysis, viewing the mining accuracy chart, running prediction algorithm on the data, and visualization. The visualization of the performance of the model is done using Lift Chat, while the model is visualized by the Model Visualization tool and Dependency Network.

This section presents the result of the models derived from the data and how the predictions were affected by the models. Each of the project's model is discussed in the order:

- JQuery (Project 1),
- Font-Awesome (Project 4),
- ReactJS (Project 6),
- Atom (Project 9).

The target prediction variable is Project Yearly LOC Churn which is the sum of all the churn Yearly. The total number of variable used for the prediction is 13 and they are listed in Table 6.2 below:

Table 6.2 List of variables used for estimations

1.	Avg Previous Imp Commits
2.	Avg Previous OO Commits
3.	Avg Previous XML Commits
4.	Avg Previous XSL Commits
5.	Developers On Project To Date
6.	Imp Developers on Project to Date
7.	Committer Previous Commits

8.	Imperative Files
9.	OO Developers On Project To Date
10.	OO Files
11.	XML Developers On Project To Date
12.	XML Files
13.	XSL Developers On Project To Date

The explanations to the variables in Table 6.2 are as follows:

- OO Files, is the count of Object-oriented programming language files in the project's given revision. The OO files considered in this works are those with the following extensions: .cpp, .cs, .php, .java, .cxx, .hpp, .js, .d, .fs, .vb, .ts, .py.
- Imperative Files, the number of Imperative files in the project's given revision. The Imperative files considered in this works are those with the following extensions: .c, .cpp, .cxx, .cs, .php, .java, .cxx, .h, .hpp, .js, .py, .rb, .d, .groovy, .fs, .fsx.
- XML Files, is the count of XML files in the project's given revision. The XML files considered in this works are those with the following extensions: .xml, .xsd, .wsdl, .xsl files.
- The Avg Previous OO Commits, is number of previous commits on OO languages files divided by the number of developers on project to date.
- Avg Previous Imp Commits, is the count of previous commits on Imperative languages files divided by the number of developers on project to date.
- Avg Previous XML Commits, is the count of previous commits on XML languages files divided by the number of developers on project to date.
- Avg Previous XSL is number of previous commits on XSL files divided by the Number of developers on project to date.
- Developers On Project To Date, are number of developers active on the project to the commit date.
- Imp Developers on Project to Date, are number of Developers whose commits make changes to imperative files on the project to the commit date.
- Committer Previous Commits, are the number of commits the contributor of the considered commit has previously made in the project.

- OO Developers On Project To Date, is the number of Developers whose commits make changes to Object-oriented programming language files on the project to the commit date.
- XML Developers On Project To Date, is the number of Developers whose commits make changes to XML files on the project to the commit date.
- XSL Developers On Project To Date, are number of Developers whose commits make changes to XSL files on the project to the commit date

The JQuery model as shown in Figure 6.2 reveals that there were 12 variables which influenced its prediction result. Most of them are more likely to affect the future behavior of this project as more revision is done on it. Selecting any of the Cases (or variables) specified would highlight the impact of the variable in the prediction and show it in the Microsoft (Decision) Tree Viewer.

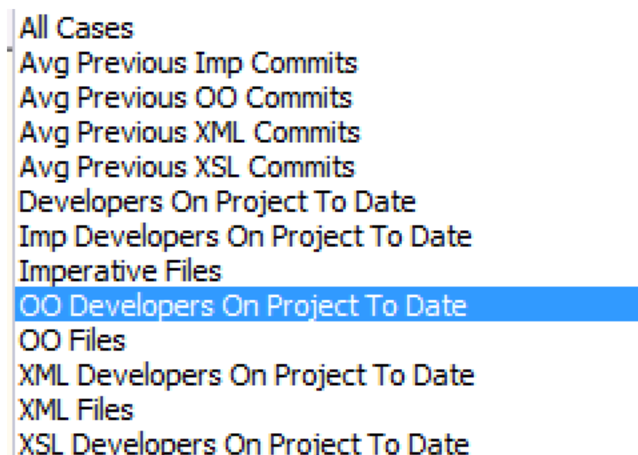


Figure 6.2 JQuery's prediction model variables used.

The commits average, the Developers count, and the Number of files (based on type) are more influential variable in the model. However, the main variable that has more influence over the prediction is "Avg Previous Imp Commits" that is, Average Previous Imperative Commits (the average count of commits in the previous revisions that touched Imperative files) as shown in the Dependency Network in Figure 6.3.

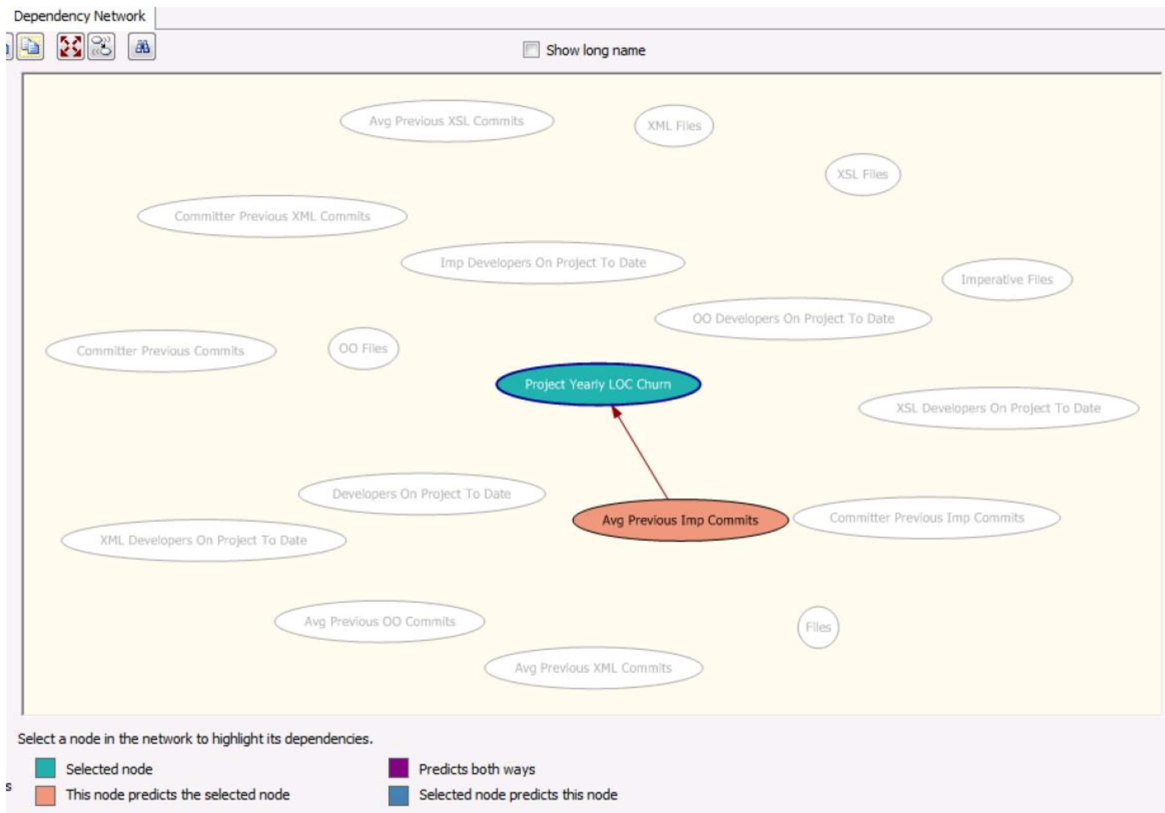


Figure 6.3 JQuery's model Dependency Network at the closest node

It could be observed that 'Avg Previous Imp Commits' predicts the 'Project Yearly LOC Churn' in the project. The implication of having that variable with more impact on the result mean that since the most important Language is JavaScript, hence the most prominent factor is the frequency of commit made on this file either by bug fixes or new feature implementation.

Font Awesome has 9 cases (or variables) that affects the prediction in general. The committer's actions, the developers on the project and the count of Files in the projects were the factors. From the Dependency Network of the most influential variable on Font-Awesome, Object-Oriented Developers on Project to date is the most influential variable of the prediction on this project.

ReactJS project's model has a total of 13 variables (or cases) that makes the prediction on the target Project Yearly LOC Churn. Out of the 13, the most influential of all the variables is the 'XML Developers on Project To Date'.

Atom's model dependency network on the other hand, has a total of **8** variables to make its prediction. The most influential of them is 'Imp Developers on Project To Date'. Given that JavaScript, the programming language used in writing Atom (IDE) is imperative in nature

although an Object-Oriented Programming (OOP) language as well, this may be a factor that led to having Imperative Developers on the Project.

6.3 Calculating the Module size of each project

Since module size depends on the date of Alternative cost and module per project, the following describes the methods or steps by which the module size calculation is done.

The module size for each project is calculated by taking a project at a time, then starting from level-1 modules, we go through all the dates.

For each of the dates and module matched per record, we check the count of each file in that module and submodules by checking out on the last commit of the given date.

The description of the method with JQuery project is as follows:

1. Given the date 10-12-2017 and the module 'jquery/src/jx'
2. Taking the last commit for that day, we *checkout* the hash string returned
3. then use a tool called line-counter[20] to count the Lines of code of each file found in that module down to submodules.
4. This count is then saved on the same record in 'fixes' column for that record.

In the count of the Lines of code, all lines in the files are part of the counts that is, comments, and even blank lines. This ensures that the result is consistent with the ones we have from the calculation of our alternative cost, because comments and blank lines were also considered as Lines of code.

6.4 Correlations

The correlations of these three metrics are calculated using the comparison of a given module against other modules (apart from that module itself). Assuming the given module is $\mathbf{M_i}$ and other modules is $\mathbf{M_oN}$ where N is any of the other module, we first find the absolute difference of the considered metric say size (in LOC) of the contrasting modules, then we calculate the comparison.

The comparison of these two modules is calculated using an *ideal* error threshold (the calculation of the error threshold is in Appendix IV. Say the Threshold value of metric Size is $\mathbf{T_s}$ so the module size difference looks as follow:

[20] https://pypi.python.org/pypi/line-counter/0.7.4

$$MsD = ABS(Mi - MoN)$$

Equation 1 Module-Size difference calculation

Using MsD (Module Size Difference), our MsC (Module Size Compare) follows the following logic:

Check:

Check: if MsD > Ts;

Then:

Check: if S1 > S2

then MsC = 1;

else MsC = -1;

Otherwise:

MsC = 0

The same logic is applied to get that of Alternative Cost and Bugs. The correlation calculation is then done on the result of these comparison. Table 6.3 shows the Threshold/Estimation model errors values obtained at different Confidence level for Module Size, Cost, and Bugs. The link to the script used for calculating the Model error is in Appendix III. While the bold values are used as threshold for comparing the values of the three metrics.

Confidence Level	Cost Error (LOC)	Bugs Error	Module Size (LOC)
0.80	1,800	2	1,658
0.90	4,200	**4**	**8,038**
0.95	**8,000**	8	25,270
0.99	44,400	52	112,858
0.999	166,000	118	267,558

Table 6.3 Alternative Cost Estimation Model Errors for all Metrics at Various Confidence Levels.

The chosen confidence level for alternative cost was 95 percent, while for both bugs and Module size at 90 percent, the three of values 8000, 4, and 8038 respectively.

After having calculated the comparison of the modules based on size, cost, and bugs we find the Kendall correlations and the result is given the subsequent sub-sections.

6.4.1 Correlations of Module Size and Bugs

This section explains the result of the correlation between the size of the modules and the number of bugs found in those modules. The higher modules in the plotted correlation graphs describes module level 2 as red, 3 as green, and 4 as blue.

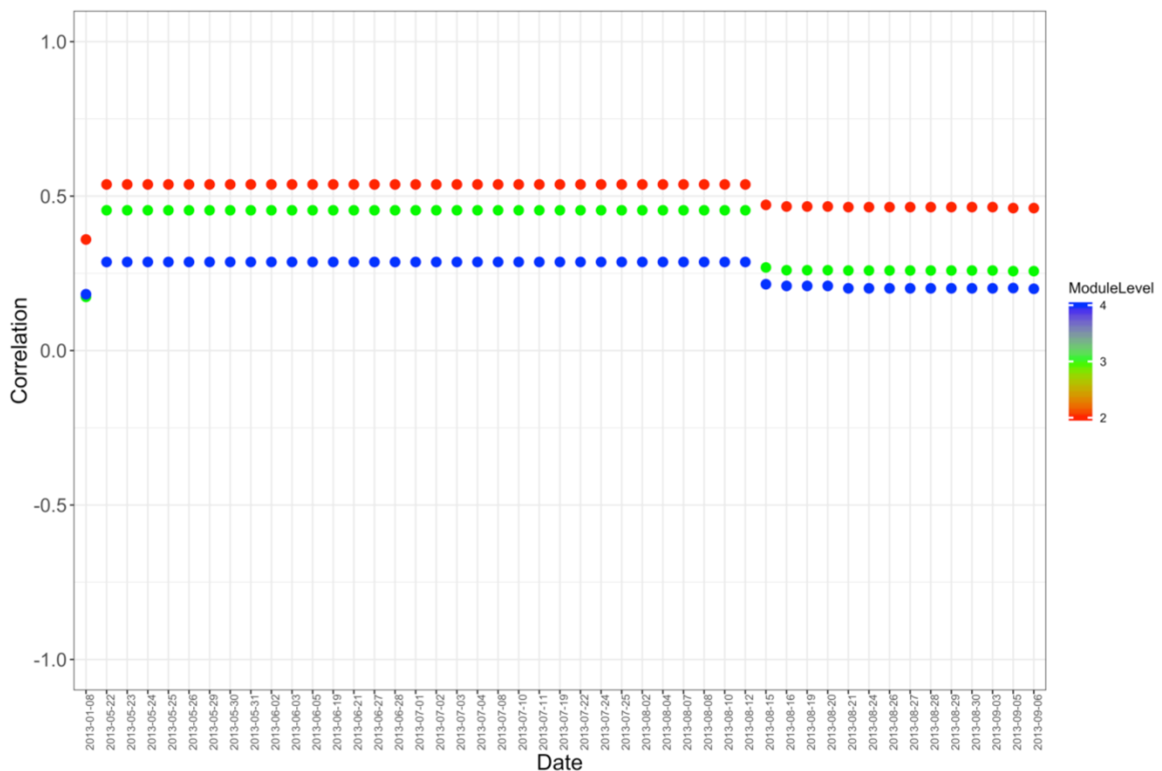


Figure 6.4 JQuery's Module Size and Bugs correlation

The correlation between the size of modules and bugs in JQuery as shown in Figure 6.4 is positive from depth level 2 to 4. The modules at level 2 has higher correlations than the others in the period while level 4 has less correlation with bug.

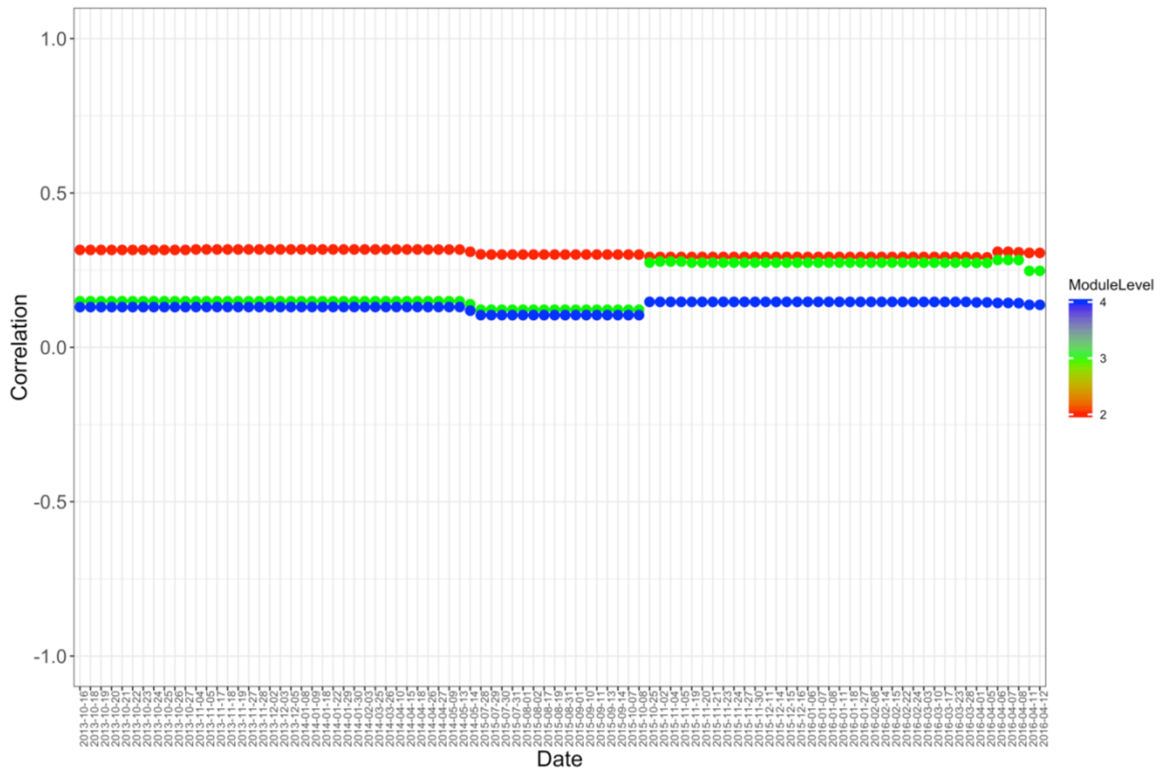


Figure 6.5 Font-Awesome's Module size and Bugs correlation

Just like JQuery, Font Awesome modules size are also positively correlated with the bugs in the modules during the time span considered. Also, the modules at level 2 shows higher correlation than others, while those at level 4 shows lower correlation as shown in Figure 6.5.

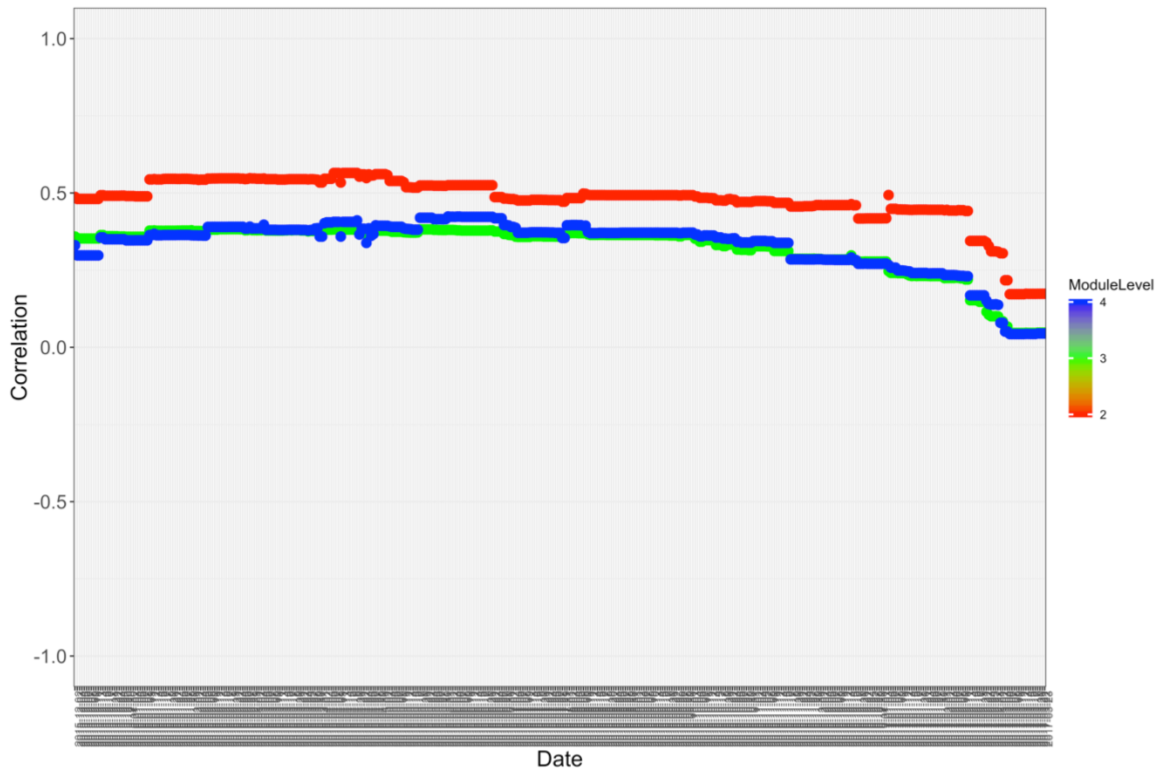


Figure 6.6 ReactJS' Module size and Bugs correlation

Figure 6.6 shows ReactJS project's module size and bugs correlation to be positive throughout and level 2 modules has higher while level 4 has lower correlation with bugs.

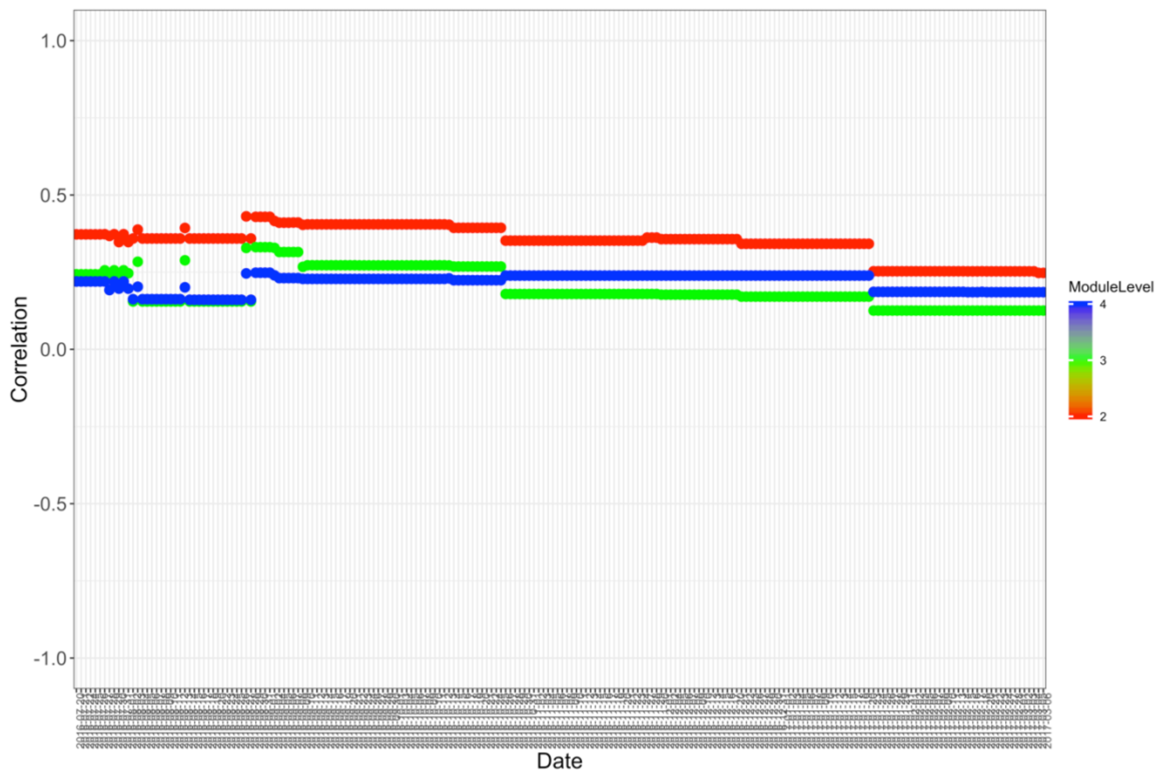


Figure 6.7 Atom's Module size and Bugs correlation

Atom's correlation graph in Figure 6.7 also shows high correlation between module size and bugs in the project. There is a common pattern in the correlation of the module size and bugs for all project: that is, they are all positively correlated at all levels where module level 2 has higher correlation than other levels for a given date. It could be inferred then that there's more possibility that the size of the module tends to the increment of bugs. This obviously is an opposition to the previously studied work (Koru, Zhang, Emam, & Liu., 2009) where they proved that under a fixed budget a company can consider low sized modules in the process of quality assurance and bugs localization.

Since we intend to observe the strength of the association of these two metrics, we consider the correlation between the modules of these four project right before finding their differences between other modules.

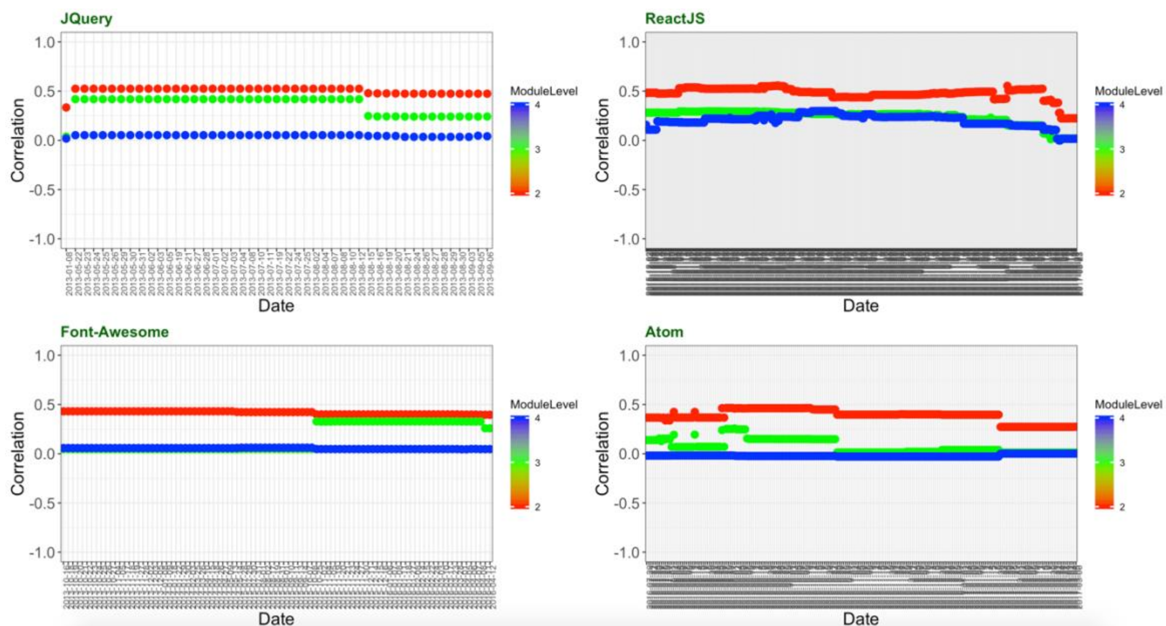


Figure 6.8 The correlation between Module (Non-Compared) Size and Bugs

Figure 6.8 shows the correlation between the size of the modules (before calculating their comparison with other modules) and bugs at the given date. The result shows that, similarly to the correlations between the compared module size and bugs, they have positive correlation, except for some period where the modules at level 4 are slightly below zero (0) and those at level 2 size have higher correlation with bugs in all the four projects. Overall, we can conclude based on our observations that the size of a module is a good indicator of the higher number of bugs, thus module size would be a good metric when performing quality assurance task.

6.4.2 Correlations of Alternative Cost and Bug fixes

The correlation of the Alternative Cost and Bug fixes for JQuery (Project 1) are all negative as shown in Figure 6.9 Also the higher the depth of the module the better the correlation. We can say, the bug fixes activity could possibly lead to the reduction in the AC of the modules.

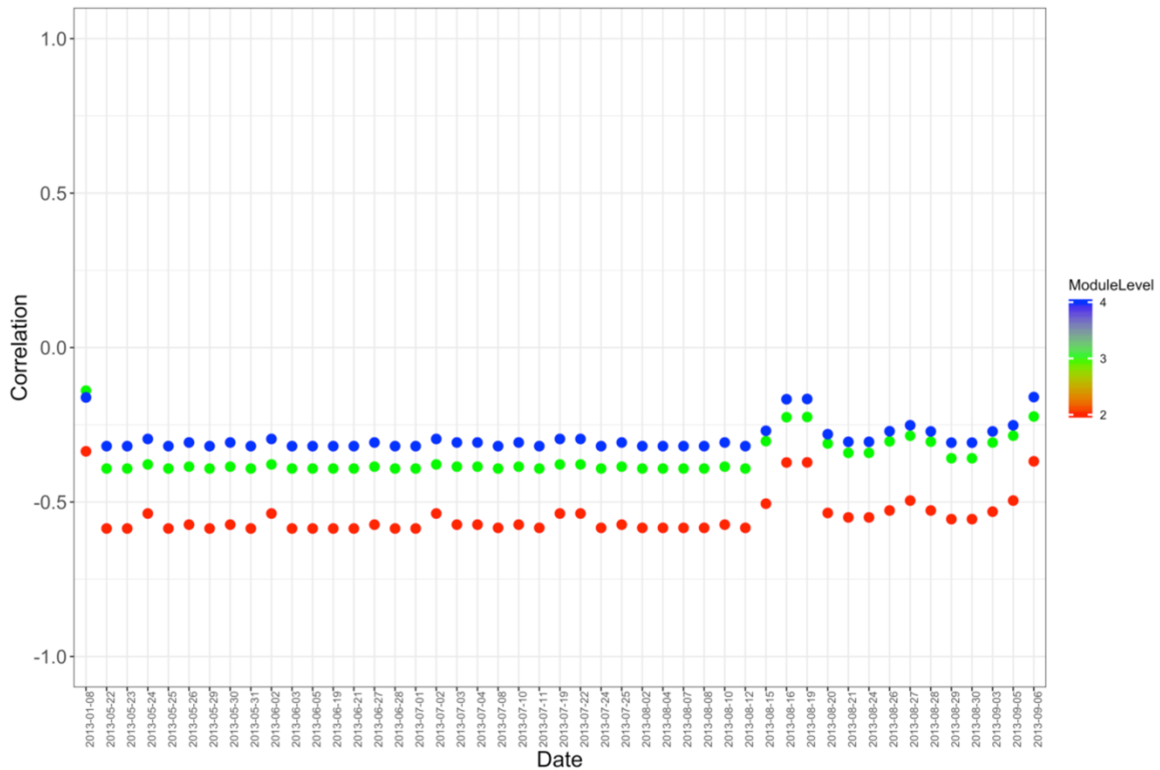


Figure 6.9 JQuery's AC and Bug fixes correlation

Unlike the Atom and JQuery projects with total negative correlation, Font-Awesome (FA) and React has positive correlations. FA has more positive correlations between 0.15 and 0.5 at all level 2,3,4 but has slightly negative correlations at -0.6 and -0.4. This is depicted in Figure 6.10.

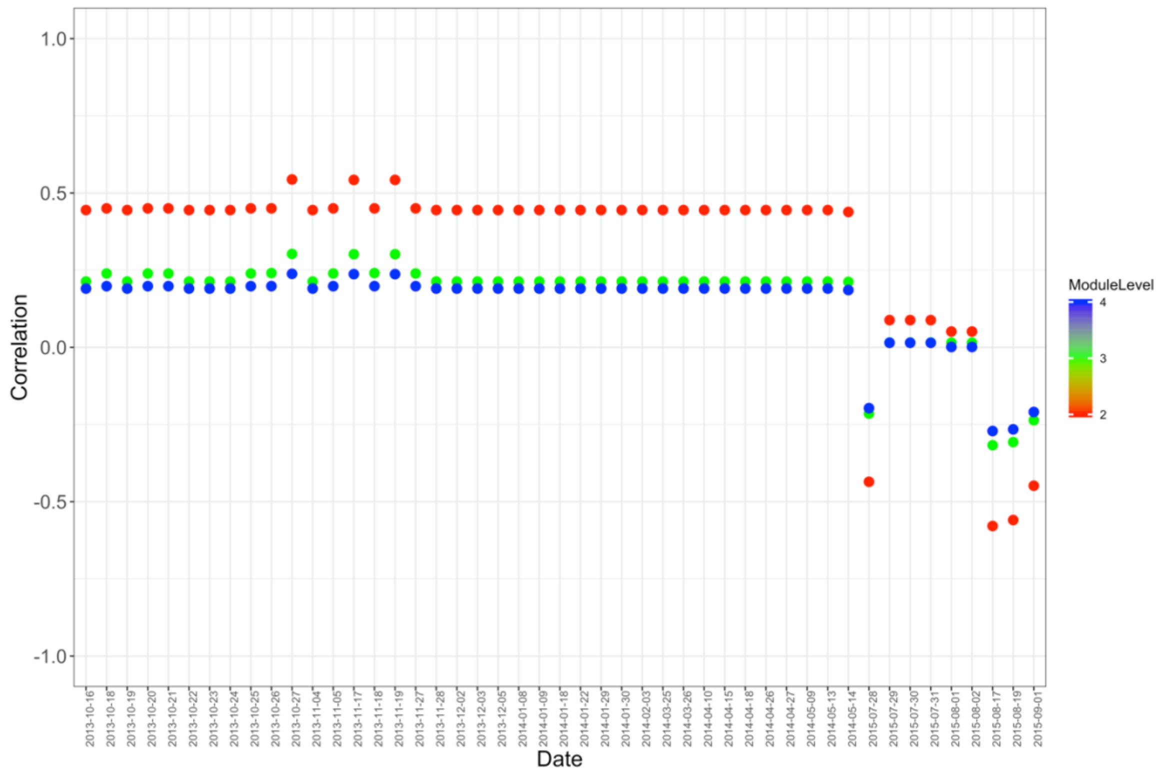


Figure 6.10 Font-Awesome’s AC and Bug fixes correlation

The exceptional correlation result is that of ReactJS Project. Figure 6.11 shows that ReactJS has all correlation between AC and Bug fixes to be positive. The outermost module level (i.e. 2) has more correlation with bug fixes, while others have lower correlation for most AC dates.

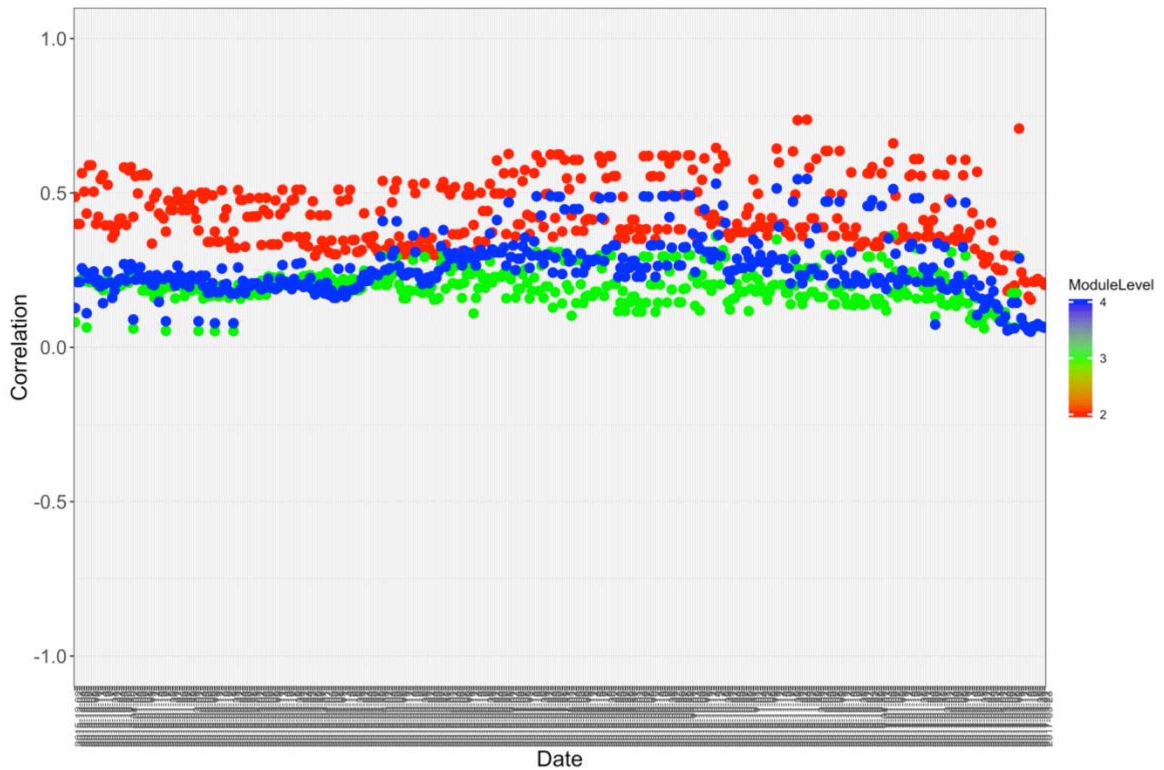


Figure 6.11 React's AC and Bug fixes correlation

The pattern of correlation observed in React project could also be found in other project, that is the outermost modules at level 2 has higher positive or negative correlation depending on the direction the correlation of the project faces. This is to say, that if the correlations in a project tends to negative the outermost modules have lower correlation, otherwise just like React they have higher correlation.

Atom just like JQuery has negative correlations for all the modules (Level 2 – 4). The notable pattern is that the deeper the module the less negative the correlation tends to, and that they share similar increase and decrease across the period considered. This is depicted in Figure 6.12

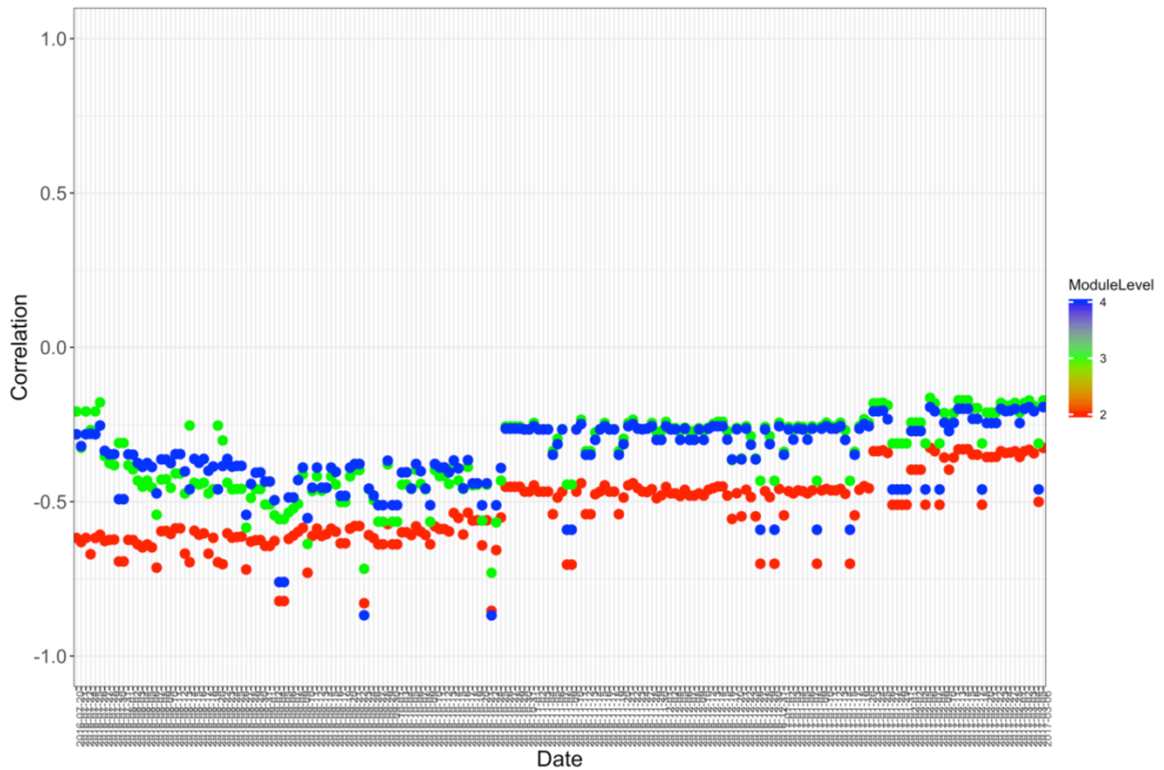


Figure 6.12 Atom's AC and Bug fixes correlation

Overall, the effect of AC on Bug seems to be similar regardless of the project and the module level. For Font-Awesome and React, the Alternative cost maintains a positive correlation with the number of bugs in their modules, while for JQuery and Atom it shows a negative correlation. The cause of this remains obscured as the factor that led to this cannot be easily observed, but it appears that Alternative Cost and bugs in module has no consistent association hence, it would not be a good metric to consider when performing a quality check or fixes on a software project.

7 Threats to Validity

In this work, the method of finding bugs goes by the PR (Pull Requests). As already described in section **Error! Reference source not found.**, Pull Requests were valid when it confirms that an issue raised was fixed. Also, we ensured that reported bugs were identified correctly and picked only when the ones which were fixed were by a Pull Request. It's been a practice in most development model that changes were made to the main development branch (Julien, 2017), so we are confident that we got the right number of bugs. One of the point of concern is, firstly a situation where this model is not followed (which is most unlikely given the negative implication of the result of direct fix on a project); secondly, there could be bugs fixed without Pull Request (that is, fixed in within a bunch of another Pull Request or not even recorded in the Version Control System); and thirdly, our method of finding bugs ensures we have limited bugs number per a given date in the modules as against finding bugs based on commits activities: a method used by (Koru, Zhang, Emam, & Liu., 2009) the implication of which ensured that our method has to be followed in order to replicate this study.

Generally, we found a moderate relationship between the size of a module and the bugs; meaning that as the module grows the bugs also increases. There could have been series of factor that influences this, for example, the pattern of development on each project therefore ensuring modules with higher size are the core of the software hence there are more activities on it. Also in finding the correlation between the size of the modules and bugs, we do not consider the complexity of the modules and the relation it has with other modules; that is, how each programming languages' classes or interfaces make use of the files in those modules. But for consistency, we measured the size of each modules against the number of lines of code, with comments and blank lines, so that we can don't have to deal with understanding the format of the languages written in each file in those modules. This may be an issue for a research into a specific programming language project or for example a minified code file, however for our case it was not important to consider those details.

Contrary to the size-bugs correlation, the AC-bugs correlation gives us the notion that there's likelihood that AC does not necessarily say anything about the number of bugs that exists or may appear in the software; this could be a cause for concern since the alternative cost calculation is derived from the code churn in the software project. Also, more importantly, due to the fact that the oddities in the AC data were not explained, the observations on AC are inconclusive (as there is a good chance of an error in the analysis).

Furthermore, in categorizing the revision data used for the projects' churn estimation to different languages group such as Object-Oriented and Imperative, we chose a restricted list of programming languages into such categories some retrieved from (Karus, 2014) and others from List of Imperative Programming languages[21] and List of Object-Oriented Programming Languages[22] both on Wikipedia[23]. The impact this has on the result though may be less significant but since there may be emergence of new languages, therefore its worth observing and upgrading the list when this work is replicated.

[21] https://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Imperative_languages
[22] https://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages
[23] https://en.wikipedia.org/wiki/Main_Page

8 Challenges

There were series of challenges encountered during the time of data collection. These challenges were not limited to mere technical ones, but most importantly finding the right data.

As initially prescribed, there were series of projects already available whose generated modules and summary had been calculated and retrieved, hence what was needed is to find out their issue repositories and use this information to answer the question that this work poses.

The few major issues faced are the following:

1. Most of the repositories have been moved: For most of the projects, even though their projects were still available online yet they were scarcely found. The process of searching out for these repositories (with their name and/or URLs) took a huge amount of time, and worst case many of them could not be found.
2. Non-availability of public (Standard) Issues Tracking System: The fact that most of the bug reports are managed through a mailing lists makes it difficult at first to even search out the issues in the system, and secondly mailing list does not seems to provide a trackable record of changes in a project. This means that, when changes are made based on proposal, the changes become difficult to link to a revision in the history of the project.
3. Some of them have switched from SVN to GIT: Most likely because of the new features in the GIT Version Control System, a good number of the projects used in the method which this thesis work set to use now are on GIT and consequently, a lot of changes in structure and how the projects are managed.
4. Few issues log/data: Some of these projects with issues data after moving their project to GitHub because they are new in the system provides only a few bug/issue data, even though the project is old. This means that we might not find the result of the analysis needed as useful as it should be.

As for the above challenges, the time taken to discover these limitations were enough to reproduce the steps of getting fresh projects with standard bug tracking system and finding out their modules then using the issues data to find out the modules that are most bug-fixed. However, after discovering that attempting to get the issues for these available projects was difficult, and because there were already **nine (9)** GIT projects whose issues data have been retrieved before this discovery, and to run the tools that were used on these old projects was

not possible because the GIT tool that was developed was not completely available and looking for another set of SVN or CVS based projects that has standard issue tracker would mean another time consumption. Therefore, the need to develop another method of retrieving those GIT based projects became inevitable to complete this work, hence, the more time was taken.

The challenges that further surfaced during the process of retrieving the data from the new projects are centered around ensuring the data retrieved from these are consistent with the data that is needed for analysis. The analysis data are stored on different server, the one we have has to be in the same format in order to have the right result when running the analysis tools. Further challenges experienced were: the extended time needed to repeat process including retrieving data from projects source, run the analysis tools, and observe the result the errors are fixed.

9 Conclusions and Contributions

We investigated the impact module size and alternative cost of a project has on the bugs that exists in it, that is to understand whether there is a relationship between them and the bugs in the system. Using the historical revision and bug data that we gathered from the four different projects that we considered in this work, we calculated the estimation of the churn using the yearly Line of Code Churn. Using these estimations for each module per given date, we find the correlation between the bugs and the derived AC, and between the size of those modules (in LOC) and the bugs in them. From our analysis, we found out that bugs is associated with the size of the modules: that it, when doing quality assurance (QA) for example bug localization activities, it would be profitable to consider modules of large size first. However, our analysis shows that using the Alternative Cost of a project is not adequate proof for bugs existence, that is, there's no justifiable relationship between AC and bugs in the project.

This work helps developers, testers, and other project's stake holders invest their resources in the right place when performing QA of their projects, and consider other metrics that would be appropriate in making decision in the QA process. We have shown how using historical data of a software project can improve the quality of a software in the future.

Furthermore, we have attempted to investigate how the AC of a software affects the different components of the software, which ensures that this work provides a ground for further research and investigation of how the using LOC as a measure of software effort estimate may be a good metric in determining some key important areas such as Software Quality of that software. This means that some of the limitations we encountered, that is, the oddities of very high AC that we couldn't find an explanation for could be resolved by investigating the errors during analysis with other methods (such as breaking the process in steps) thus resolving those errors and consequently having a more conclusive result.

10 Acknowledgement

Special thanks to God Almighty for giving me the privilege to be alive in the course this thesis and for giving me peace and salvation that comes from His amazing grace only through Jesus. Special thanks to my supervisor Siim Karus for his attention, time, thorough guidance and feedbacks throughout, it would be impossible to have completed this thesis without you. Thanks also to my colleagues at school and work for their support and consideration when I needed most. Special appreciation goes to my family entirely, my parents Mr. and Mrs. Omisakin, and my siblings for their prayers and constant check on my progress. To all my friends especially very close friends, in Isaac Agaba, Darwin Sivaligapandi, Fortunat Mutunda and my dear sister Ifeoluwaposimi Oluwafemi for their unrelenting encouragements. To Ivan Ojiambo, for proofreading my thesis and giving critical feedbacks as much as he could do in the short time he has. To the entire Tartu International Fellowship Family for their spiritual support every time. I will be grateful always to my wife-to-be, Joy Omoshola Akande for encouraging and cheering for me. Finally, to the University of Tartu for giving to me this platform to advance in my academic career, I will always be grateful.

11 References

Artemij, F. (2015). *React.js Essentials.* BirminghamB3 2PB, UK: Packt Publishing Ltd.

Atlassian Tutorial - What is Git. . (n.d.). Retrieved 05 15, 2018, from Atlassian: https://www.atlassian.com/git/tutorials/what-is-git

Balzer, R. (1985). A 15 Year Perspective on Automatic Programming. IEEE.

Bell, R. M., Ostrand, T. J., & Weyuker, E. J. (2006). Looking For Bugs in All the Right Places. *ISSTA '06 Proceedings of the 2006 international symposium on Software testing and analysis* (pp. Pages 61-72). New York, NY: ACM.

Bozzon, A., Comai, S., Fraternali, P., & Carughi, G. T. (2005). Conceptual modeling and code generation for rich internet applications. *Proceedings of the 6th international conference on Web engineering* (pp. 353-360). ACM.

Buchanan, J. M. (1991). Opportunity Cost. In J. M. Buchanan, *The World of Economics.* London: Palgrave Macmillan.

Denney, E., & Fischer, B. (2006). A generic annotation inference algorithm for the safety certification of automatically generated code. *Proceedings of the 5th international conference on Generative programming and component engineering* (pp. 121-130). ACM.

Font-Awesome's website . (n.d.). Retrieved 05 15, 2018, from https://fontawesome.com/v4.7.0/

GitStats Repository. (n.d.). Retrieved 05 15, 2018, from https://github.com/dmitryn/GitStats

Hansson, D. H. (Ed.). (2016, July 07). *Sofware has bugs. This is normal.* (Basecamp) Retrieved May 26, 2018, from signalvnoise: https://m.signalvnoise.com/software-has-bugs-this-is-normal-f64761a262ca

Homepage of Atom. (n.d.). Retrieved 02 2018, 12, from https://atom.io

Homepage of React. (n.d.). Retrieved 05 15, 2018, from React: https://facebook.github.io/react/

James, L. (n.d.). *Correlation (Pearson, Kendall, Spearman)*. Retrieved August 2, 2018, from Statistics Solutions: http://www.statisticssolutions.com/correlation-pearson-kendall-spearman/

Jones, J. A., Harrold, M. J., & Stasko, J. T. (2001). Visualization for fault localization. *Proceedings of ICSE 2001 Workshop on Software Visualization, Toronto, Ontario, Canada*, (pp. 71-75).

jQuery Homepage. (n.d.). Retrieved 05 15, 2018, from jQuery: http://jquery.com/

Julien, D. (2017, January 30). *Blog: Collaborative Development Model With GIT Using Pull Requests*. (D. Julien, Editor) Retrieved July 29, 2018, from Julien Dubreuil - Freelance spécialisé Drupal, architecte web et développeur Drupal - ScrumMaster: https://juliendubreuil.fr/blog/development/collaborative-development-model-git-pull-requests/

Kameyama, Y., Oleg, K., & Shan, C.-c. (2015). Combinators for impure yet hygienic code generation. *112*, 120-144. Science of Computer Programming.

Karus, S. (2014). Detecting Modules with Computer-Generated Code.

Koru, A. G., & Emam, K. E. (2009). The Theory of Relative Dependency: Higher Coupling Concentration in Smaller Modules and its Implications for Software Refactoring and Quality. *IEEE Software, 27*(2), 81-89.

Koru, A. G., Zhang, D., Emam, K. E., & Liu., H. (2009). An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules. *IEEE Transactions on Software Engineering. 35*, pp. 293-304. IEEE.

Laravel's homepage. (n.d.). Retrieved 05 15, 2018, from Laravel: https://laravel.com/

Lechanceux, L., & Eugene, S. (2017). Comparison of the expressiveness and performance of template-based code generation tools. *SLE 2017 Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering* (pp. 206-216). New York, NY: ACM.

Malaiya, Y. K., & Denton, J. (2000). Module size distribution and defect density. *Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000* (pp. 62-71). San Jose, CA: IEEE.

Mozilla Firefox. (2018, July 29). *What software do I need to build a website?* Retrieved August 4, 2018, from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_software_do_I_need#Creating_and_editing_webpages

Nachiappan, N., & Ball, T. (2005). Use of Relative Code Churn Measures to Predict System Defect Density. *Proceedings of the 27th international conference on Software engineering* (pp. pp. 284-292). ACM.

Ostrand, T. J., Weyuker, E. J., & Bell, R. M. (2004). Where the Bugs Are. *ISSTA '04 Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis. Volume 29*, pp. 86-96. Boston, Massachusetts: ACM SIGSOFT Software Engineering Notes.

Peter, W. (2013, November 18). *10 reasons the browser is becoming the universal OS*. Retrieved August 4, 2018, from InfoWorld: https://www.infoworld.com/article/2609165/web-browsers/10-reasons-the-browser-is-becoming-the-universal-os.html

Peyrott, S. (2017, January 16). *Auth0: A Brief History of JavaScript*. Retrieved August 9, 2018, from Auth0: https://auth0.com/blog/a-brief-history-of-javascript/

Rosenberg, J. (1997). Some misconceptions about lines of code. *Software Metrics Symposium, 1997. Proceedings., Fourth International* (pp. pp. 137-142). Albuquerque, NM: IEEE.

Roychoudhury, A. (2010). Debugging as a Science, that too, when your Program is Changing. *Electronic Notes in Theoretical Computer Science*, 3-15.

Stürmer, I., Weinberg, D., & Conrad, M. (2005). Overview of existing safeguarding techniques for automatically generated code. *ACM SIGSOFT Software Engineering Notes, 30*(4).

Sturm, T., Voss, J. v., & Boger, a. M. (2002). Generating code from UML with velocity templates. *International Conference on the Unified Modeling Language.* Berlin Heidelberg: Springer.

Thorsten, S., Voss, J. v., & Boger, M. (2002). Generating code from UML with velocity templates. *International Conference on the Unified Modeling Language.* Berlin Heidelberg.

Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2009). A survey on software fault localization.

Yasmeen, S. (2014). Software Bug Detection Algorithm using Data mining Techniques. *International Journal of Innovative Research in Advanced Engineering (IJIRAE), 1*(5).

Yinhuan, Z., Beizhan, W., & Yilong, Z. (2009). Estimation of software projects effort based on function point. *2009 4th International Conference on Computer Science & Education.* Nanning, China: IEEE.

Appendix

I. VCS_Modules Table Structure

```
create table VCS_Modules
(
  ModuleDateRevisionId  bigint unsigned auto_increment
    primary key,
  ProjectDateRevisionId bigint unsigned not null,
  Date                  datetime        not null,
  ProjectId             int unsigned    not null,
  CommitId              bigint unsigned not null,
  ModuleId              varchar(255)    not null,
  Files                 int unsigned    not null,
  XMLFiles              int unsigned    not null,
  XLSFiles              int unsigned    not null,
  OOFiles               int unsigned    not null,
  ImperativeFiles       int unsigned    not null,
  JavaFiles             int unsigned    not null,
  CPPFiles              int unsigned    not null,
  CFiles                int unsigned    not null,
  CSharpFiles           int unsigned    not null,
  PHPFiles              int unsigned    not null,
  JavaScriptFiles       int unsigned    not null,
  RubyFiles             int unsigned    not null,
  ModulePath            varchar(255)    not null,
  created_at            timestamp       null,
  updated_at            timestamp       null,
  constraint
vcs_modules_projectid_projectdaterevisionid_modulepath_unique
  unique (ProjectId, ProjectDateRevisionId, ModulePath)
)
  collate = utf8_unicode_ci;
```

II. VCSEstimations Table Structure

```
-- auto-generated definition
create table VCSEstimations
(
  Id                                bigint unsigned auto_increment
    primary key,
  ProjectId                         bigint unsigned          not null,
  ProjectDateRevisionId             bigint                   not null,
  Date                              datetime                 not null,
  RevisionNumber                    int unsigned             not null,
  Avg_Previous_Imp_Commits          decimal(38, 6)           null,
  Avg_Previous_OO_Commits           decimal(38, 6)           null,
  Avg_Previous_XML_Commits          decimal(38, 6)           null,
  Avg_Previous_XSL_Commits          decimal(38, 6)           null,
  Committer_Previous_Commits        int unsigned             null,
  Committer_Previous_Imp_Commits    int unsigned             null,
  Committer_Previous_OO_Commits     int unsigned             null,
  Committer_Previous_XML_Commits    int unsigned             null,
  Committer_Previous_XSL_Commits    int unsigned             null,
  Developers_On_Project_To_Date     int unsigned             null,
  Imp_Developers_On_Project_To_Date int unsigned             null,
  Files                             int unsigned default '0' not null,
  Imperative_Files                  int unsigned             null,
  OO_Developers_On_Project_To_Date  int unsigned             null,
  OO_Files                          int unsigned             null,
  Total_Developers                  int unsigned             null,
  Total_Imp_Developers              int unsigned             null,
  Total_OO_Developers               int unsigned             null,
  Total_XML_Developers              int unsigned             null,
  Total_XSL_Developers              int                      null,
  XML_Developers_On_Project_To_Date int unsigned             null,
  XML_Files                         int unsigned             null,
  XSL_Developers_On_Project_To_Date int unsigned             null,
  XSL_Files                         int unsigned             null,
  DevelopmentStageAsPercent         varchar(255)             null,
  created_at                        timestamp                null,
  updated_at                        timestamp                null,
  ProjectYearlyLOCChurn             bigint default '0'       not null,
  constraint project_date_index
  unique (ProjectId, ProjectDateRevisionId)
)
  collate = utf8_unicode_ci;
```

III. Project Resource link

- The backend repo: https://github.com/omitobi/issuesminer
- The front-end repo: https://github.com/omitobi/issuesminer_front
- R script for visualization and correlation: https://github.com/omitobi/issuesminerscripts

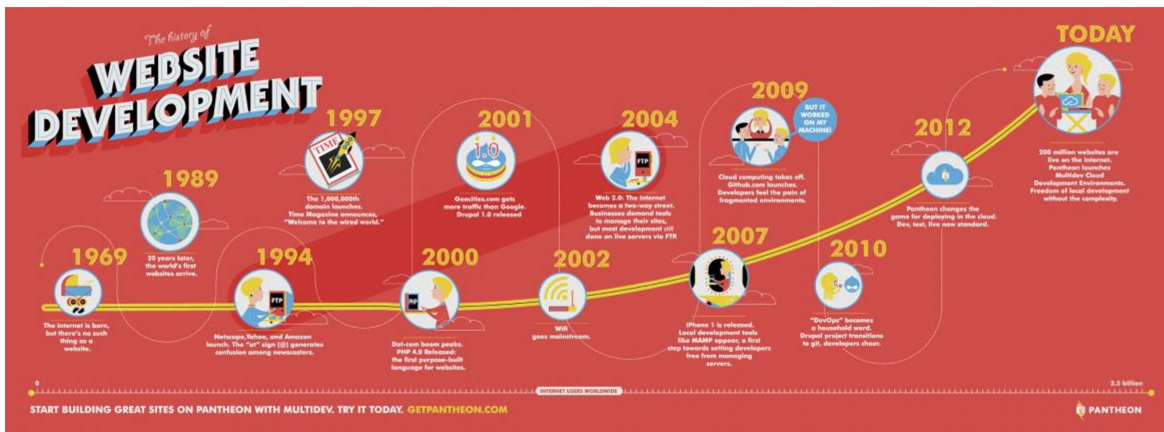
IV. Script to calculate Threshold

```
setwd("/Users/OMITOBISAM/Desktop/Thesis Data Import/Progress_6_Dec")

library('dplyr')
library('ggplot2')
library(RMySQL)

#-------------------------------------
#               Examples
#----------------------------------------------------------------------
------------
# by fixesDifference ASC limit 1 offset 24354918

# rs = dbSendQuery(mydb, "select count(*) from projectcostdifference
where fixesDifference <= 2")

#----------------------------------------------------


# -------------------------
mydb = dbConnect(MySQL(), user='root', password='Sch..l1234',
dbname='issuesminer', host='localhost')
# -------------------------
pid_ = 1;

#--------------------------
rs_1 = dbSendQuery(mydb,'select count(Id) from projectcostdifference')
count_of_record = fetch(rs_1, n=-1)
count_of_record #==25636756
count_of_record <- 25636756

#------------------
mid_position = count_of_record/2
mid_position #=== 12818378

rs_2 = dbSendQuery(mydb,'select max(locDifference) from
projectcostdifference order by locDifference')
max_locDiff = fetch(rs_2, n=-1)
max_locDiff #===277975
max_locDiff <- 277975
#-----------------

rs_3 = dbSendQuery(mydb,'select locDifference from
projectcostdifference order by locDifference limit 1 offset 12818378')
mid_locDiff = fetch(rs_3, n=-1) #mid_locDiff
mid_locDiff #=== 62
#-----------------

rs_4 = dbSendQuery(mydb,'select count(*) from projectcostdifference
where fixesDifference<62')
count_lessthanmid = fetch(rs_4, n=-1)

count_lessthanmid#==25460908
count_lessthanmid = 25460908
#----------------------------
ninety_perc = check_at_percent(round(percent_threshold(90))); #at
90_percent = 8038
#-------
```

```
ninety5_perc = check_at_percent(round(percent_threshold(95))); #at
95_percent = 25270

ninety5_perc = check_at_percent(round(percent_threshold(99))); #at
95_percent = 25270
ninety5_perc

#-------

count_lessthanmid/count_of_record

(ninety5_perc/max_locDiff) *100
#------------

#--------Experiments-------
total__ = get_total_records(); #25636756
perc__total = round(percent_threshold(85, total_record=total__))
perc__ = check_at_percent(perc__total)
perc__

#LocDifferences compare
#|--perc--|--value--|
#| .80**  |  1658   |
#-------------------|
#| .85    |  2782   |
#-------------------|
#| .90*   |  8038   |
#-------------------|
#| .95    | 25270   |
#-------------------|
#| .99    | 112858  |
#|------------------|
#| .999   | 267558  |
#|------------------|

from_value = check_from_value(value = 8038, operator = '<=')
from_value
from_value/total__
#--------/Experiments------

#--- Method ------

get_total_records = function(table ='projectcostdifference', field =
'Id') {
  rss = dbSendQuery(mydb, paste("select count(", field,") from
",table))
  return (fetch(rss, n=-1))
}

percent_threshold = function(percent = 90, total_record) {
  return ((percent/100) * total_record);
}

check_at_percent = function(at_value, field = 'locDifference') {
  rs_s = dbSendQuery(mydb, paste("select ", field," from
projectcostdifference order by ",field," ASC limit 1 offset ",
at_value))
  return (fetch(rs_s, n=-1))
}
```

```
check_from_value = function(value, operator = '=', field =
'locDifference') {
  rs_ss = dbSendQuery(mydb, paste("select count(Id) from
projectcostdifference where ",field, " ",operator," ", value))
  return (fetch(rs_ss, n=-1))
}

#---- /Method -----
```

V. Website Development History



Source: https://pantheon.io/resources/history-web-development at 02-08-2018

VI. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **OLUWATOBI SAMUEL OMISAKIN**,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Relationship between Module Size, Alternative Cost and Bugs,

(title of thesis)

supervised by Siim Karus,

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **12.08.2018**