Bachelor Thesis Project

# A Study on the Relation between Software Quality & Test Quantity

*Author:* Adham El-Ahmad
*Supervisor:* Dr. Rüdiger Lincke
*Semester:* VT 2016
*Subject:* Computer Science

# Abstract

Testing is an essential phase of software development process. It helps to ensure the quality of the software by finding bugs, errors, and defects that may occur during the development or the deployment of the system. IT companies and field workers, spend a lot of efforts on testing a software. However, how far should testers go with testing? In this research, we study 80 open source real world projects, and explore how much testing code has been written in those projects, in comparison with production code. We also try to find if there is any relation between different project factors, such as: project size, number of contributors and the amount of testing that has been done in those projects. We also give some recommendations to help field workers determine the amount of testing needed for a software.



Keywords: Software Testing, Test Automating, Software Quality, JAVA, Open Source.

# Contents

**Glossary**

NIST: National Institute of Standard and Technology
CLOC/T: Count Line of Code/Test
LOC: Line of Code
CSV: Comma Separated Values
UAT: User Acceptance Testing
SDLC: Software Development life cycle.

# 1 Introduction

This thesis project aims to find a relation between the test quantity and software quality. Testing is important, but it also requires much effort and resources during the development process, which could lead to longer development time, increased costs and later delivery of the software to the customer. Thus, a threshold/recommendation helping to determine how much test code should be written could help to determine the right testing level for optimizing costs vs. benefits.

## 1.1 Background

Software development goes through a cycle of different phases. A software is planned, designed, deployed, fixed and then released. In order to be able to use any software without any failures, bugs, and errors, software must be tested.

Software errors cause a huge amount of problems all over the world every year. Some of these errors may cause minor problems. But for example, flight control or medical equipment systems, are not supposed to fail due to programming errors. Untested software might lead to unwanted results or even to disasters. The world still remembers the failure of the rocket Ariane 5 which has exploded only about 40 seconds of the flight initiation, where the failure turned out to be caused by a software error [2].

Bugs and errors are extremely expensive. A study made by the National Institute of Standards and Technology (NIST), found that software bugs cost the US $59.5 billion every year. The study claims that more than a third of that amount, $22.2 billion, could be saved by improved testing [8].

It is well known that the cost of fixing errors and defects increases as time goes on during the Software Development Life Cycle (SDLC) (see Figure 1). A research conducted by NASA shows that a requirement error that might cost 1 unit to fix at the requirement phase, might cost more than 1500 units at the operation phase [9]. Therefore, it is recommended to start testing as early as possible during the SDLC of the project.

Testing can be done using various software testing techniques and methodologies like white box, black box, functional testing, stress testing and User Acceptance Testing (UAT).

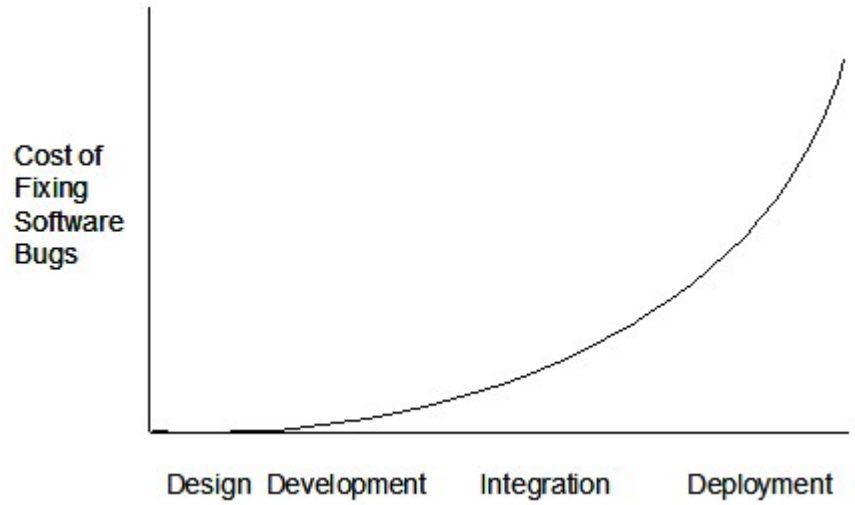


*Figure 1: Shows how the cost of fixing a bug escalates during different SDLC phases [13].*

## 1.2 Related Work

Due to the fact that software testing process is expensive, researchers and field workers have always been interested in finding a balance between testing too much and testing too little. Therefore, much research has been done aiming to find satisfying answers and effective solutions for this problem, or to find whether there is any relation between the amount needed of testing, and different projects characteristics.

In 2013, in a research conducted on open source projects, Pavneet Singh Kochhar along with Tegawende F.Biss, David Lo and Lingxiao Jiang, found that projects having test cases are bigger in size than projects without test cases. They have also found a weak correlation between numbers of developers and test cases [4].

During the same year, another research did analyze the prevalence of test cases in open source projects and the correlations between the test cases and other important software metrics. The research made the following findings: Projects with test cases have more Line of Code (LOC) than those without test cases, projects with more number of developers have more test cases, a weak positive relationship between number of test cases and number of bugs, number of test cases has a weak correlation with the number of bug reporters. The research also showed that projects written in popular languages, such as C++ ANSI C, and PHP, have higher mean numbers of test cases per project in comparison with projects in other languages [12].

Anju Bansal did conduct a research in 2014 to assess the relation between quantity of test cases and amount of code coverage. The research has shown that there is no relationship between the number of test cases and the amount of code coverage. “Code coverage does not depend on quantity of test cases; it actually depends on quality of test cases” says Anju [11].

## 1.3 Problem Formulation

Theoretically, it is well known that more testing is better, no test code is bad. Yet, to the best of our knowledge, no data suggesting a reasonable ratio between test and production code exists. I.e., it is unclear if real world projects vary with respects to the percentage of test code written for a project, and if so which distributions they follow. Thus, software developers do not know how much test code is reasonable. In particular, when discussing required quality levels with customers having project costs in mind.

## 1.4 Motivation

The cost of software testing is really high, both in time and effort. In fact, earlier studies estimated that testing can consume fifty percent, or even more, of the development costs [3]. Moreover, too much testing may lead to a delay in releasing the system for operational use, which means additional costs. This suggests a reduction in testing time [1]. On the other hand, an early release of the system without enough testing may lead to unwanted consequences. According to Amrit L. Goel & Kazuhira Okumoto, the cost of fixing a bug after release may reach a level where we favor not to use the system at all [1].

Unfortunately, up to date, there is no such a way that helps to determine the quantity of test code needed for a software to be called "Sufficiently Tested" or "High Quality Software," neither having 100% code coverage can do that. This thesis aims to suggest a satisfying answer for that, or in other words; to find a balance between test quantity and software quality, based on statistics taken from real world software projects.

## 1.5 Research Questions

To get an answer to the above problem, we plan to answer the following research questions, by assessing a sufficiently large sample of open source projects.

| | |
|---|---|
| **RQ1** | Do real world projects vary with respect to the percentage of test code written for a project? If yes, which distributions do they follow? |
| **RQ2** | Which classifiers can be used to distinguish the distributions? |
| **RQ3** | Are there thresholds/ranges for projects of a certain classification which can be used for recommendations? E.g., and open source java back end project with 5 developers and 500K lines of code and a user base > 10000 should have at least 50% of test code to be considered mature. |

By the end of this project, we assume that we will answer the above research questions. We also assume to find a variation in test code quantity between real world projects and their distributions. In addition to all of that, we assume that we will be able to give recommendations for the quantity of test needed for a software to be considered mature.

## 1.6 Scope/Limitation

We decide in our research to focus and to limit our data set to real world projects, which have Java as a main programming language, and are open source (to get access to the source code). We also only examine and consider the Java code in those projects as it is the dominant language used.

In order to detect test files in our data set, we use a heuristic approach, i.e., we consider the files whose path name contains the word: "test" as test files only.

## 1.7 Target Group

We hope that our research results will prove useful to a large audience in software development including: Developers, Testers, Software Architects, Project Managers, IT Consultants, Researchers, Sales Representatives, Business Owners, and of course Customers or Product Owners, when discussing the testing level.

## 1.8 Outline

This thesis report will be constructed as the following:

Section 2 – Method will describe the approach taken to perform this thesis research.
Section 3 – Implementation represents in detail the steps taken during this work, as well as the dataset resources and the dataset collection.
Section 4 – Results shows the results of this research, by answering the research questions.
Section 5 – Analysis an analyzation of the results, and a comparison between the results and our own thoughts.
Section 6 – Discussion a discussion of our findings and whether our research questions were answered or not.
Section 7 – Conclusion a wrap up of the thesis by writing giving a final conclusion and recommendations, as well as suggestions for any future work.

# 2 Method

*"Every discourse, even a poetic or oracular sentence, carries with it a system of rules for producing analogous things and thus an outline of methodology" - Jacques Derrida*

This section will clarify the methods used during this work, it will outline the scientific approach that has been used, and it will describe why we chose to do it that way.

## 2.1 Scientific Approach

We use exploratory research methodologies to get a feeling what data is available and how we can use it. With the results, we get a clearer understanding of the problem and do refine our research questions. We also collect quantitative data from our dataset after being analyzed. Along with some literature review, the outcomes of the analyzed data, are documented and used to answer the research questions.

## 2.2 Method Description

After taking a look at the research questions, it has been realized that they are open ended and that no previous studies have been done to study them, it has also been noticed that this research will be a groundwork for further studies. Thus, exploratory research concepts have been applied to the research questions to get them explored and to have a clear design for the whole research. This has revealed the need of a CLOC/T (Count Line of Code/Test) tool that helps in studying and analyzing the dataset.

Moreover, running the tool mentioned above results in a quantitative data, which are then saved into a CSV (Comma Separated Values). This data is collected and statistically analyzed to achieve what is needed to answer the research questions. In addition to all of that, a literature review has been done to help in documenting the results.

## 2.3 Reliability and Validity

**External validity** is related to the generalizability of our results. Despite that our dataset consists of huge projects with millions of lines of code, the results may not represent all real world projects. In addition to all of that, our study is conducted on open source projects. Thus, the results may differ when it comes to closed source projects. Our heuristic way for identifying test code might need adjustment for other projects. All the projects in our dataset are multi-language projects, we consider only the main language, i.e., the language with the highest number of lines of code in a project, which is Java for all projects in our dataset. So, we do not really know if including all the languages in the projects will give different results. Although that we have tried to ensure quality of our dataset, our results and our work, by examining that we take the right number of test files into account, the quality of our dataset might still be an issue here. In our research, we look up for test files using heuristics, i.e., we consider the files whose pathname contains the word test as test files. This might not detect test files whose pathname does not have word test or vice versa, detect some files whose path contains the word test but actually are not test files. We use that way in order to be able to handle a large number of files. However, in some cases, we manually check and count test files for some of the projects to validate results.

Threats to **internal validity** refers to whether an experimental condition makes a difference or not. We design our experiment carefully and document all required steps. The input data, used software etc. is documented and provided with this thesis. It should not be a problem for other researchers to repeat this study and come to similar results, regardless if they use different computer systems, etc. taking into considerations the small changes that should be applied on pathnames to be compatible with other computer systems, e.g., Linux.

**Reliability** ensures that the operation of a study – such as the data collection or the data analyzing procedures – can be repeated at any time yielding the same results. The reliability of a case study is important. It shall allow a later researcher to get the same results when following the same procedure. We document all important steps and decisions as well as the procedures needed to perform the data analyzing. We include projects names used in the study under Appendix section.

# 3 Implementation

This chapter gives details about our dataset and about our CLOC/T tool, as well as platforms and frameworks that have been used during this project.

## 3.1 Dataset Resources

For our research, we analyze real world projects founded online using the Black Duck Open Hub[1]. After finding the targeted project and making sure that it has a suitable size, and a certified license we then look for its repository on GitHub[2] and make sure to download the last stable version before putting it under assessment.

## 3.2 Dataset Overview

Our dataset consists of 80 projects of different sizes and types such as: frameworks, libraries, search engines, management systems and so on, with a total of 38.146.906 lines of source code, and 16.036.372 lines of test code (ca. 42%). Each of them has a well-known license like: Apache 2.0, LGPL, MIT etc.

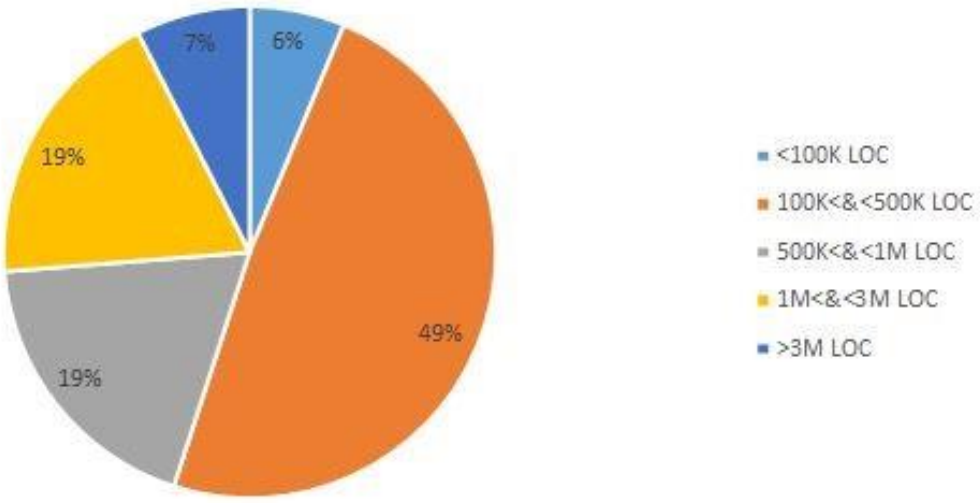


*Figure 2: Distribution of Projects in Terms of their Size*

[1] www.openhub.net The Black Duck Open Hub is an online community and public directory of free and open source software (FOSS) [5].

[2] github.com GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) [6].

The following table shows a sample of the information that has been gathered and used during our research:

| Project name | Contributors | Size | Version | Type | License |
|---|---|---|---|---|---|
| Apache Camel | 274 | 1.11MB | 2.17.0 | Framework | Apache 2.0 |
| Neo4j | 232 | 898KB | 3.0.0 | DBMS | AGPL3.0 + GPL3.0 |
| Mockito | 88 | 42KB | 1.10.19 | Library | MIT |

*Table 3.1: Sample of Projects in Our Dataset*

## 3.3 The CLOC/T Tool

A CLOC/T tool has been developed specially for this project, while keeping in mind that the tool will be used for further studies later on. Our tool has the ability to count the following:

- Total Lines of Code: All lines in each file with (.java) extension in the targeted project.
- Executable lines: Lines ending with a Semicolon ";".
- Trivial Lines: Lines that only have { or }.
- Empty Lines: Lines that contain no visible characters.
- Lines of Comments: Lines starting with // or /* or *.
- Source Code Comments Percentage: Percentage of comments in source code.
- Test Code Comments Percentage: Percentage of comments in test code.
- Code Complexity: Quantitative cyclomatic complexity measurement technique has been applied here [7]. (Applied only on source code classes)
- Number of Files: Number of the files with (.java) extension in the project.
- Class analyzes: The tool is able to analyze each class by itself and print the results directly to the console. The tool gives the user the ability to choose whether to include this feature to his work or not. (Applied only on source code classes, can be enhanced to be applied on both source and test code).

All the features named above, are applied separately on both source and test code classes, unless mentioned.

Final results are being saved to a (CSV: Comma-separated values). The file will be created automatically after each time the user runs the tool. However, if the user would like to run the tool again, it's necessary to delete the file(s) and let the tool create a new one. Otherwise, a risk of getting duplicated results might exist.

In order for the tool to run as intended, two important steps must be followed:

1. The source code of the targeted projects should be placed in one folder. The path of that folder should be passed as a parameter to the main method.

2. Targeted projects should pursue the following name convention:

```
Project Name-Type-License-Size-# of Contributors-
Version-Download Date
```

All this information can be found for each project using our data resources. The name convention can be changed if needed.

After following the steps mentioned above, the tool will be ready to work and run as intended. The tool will automatically traverse over the projects located in the specified folder.

# 4 Results

In this chapter, the results from the performed data analysis using the tool created are presented. Statistics that have been gathered during the research will be shown and briefly explained.

## 4.1 Numerical Data

The outcomes of analyzing the projects in our dataset are just numbers that show the following in each project: Total Lines of Code, Executable lines, lines of comments and the rest of features mentioned in chapter 3. The following table shows an example of our outcomes.

| Project Name | Total Lines of Code | Executable Lines | lines of Comments | Trivial Lines | Empty Lines | Code Complexity | Number of Files | Average File Complexity | Total Comment Percentage | Source Code Percentage | Total Lines of Test Code | Executable Test Lines | lines of Comments in Tests | Trivial Lines in Tests | Empty Lines in Tests | Number of Test Files | Comment Percentage in Test | Test Code Percentage |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Activiti | 215098 | 104765 | 49150 | 25793 | 35390 | 39154.0 | 2134 | 18.347704 | 22% | 67% | 102110 | 60797 | 15748 | 7039 | 18526 | 755 | 15% | 32% |
| ApacheActiveMQ | 355331 | 160352 | 102229 | 41372 | 51378 | 74633.0 | 2199 | 33.939518 | 28% | 55% | 285299 | 160275 | 52331 | 25501 | 47192 | 1987 | 18% | 44% |
| ApacheAxiom | 77229 | 30590 | 29738 | 8173 | 8728 | 15852.0 | 834 | 19.007195 | 38% | 49% | 79018 | 38019 | 26443 | 6494 | 8062 | 1187 | 33% | 50% |
| ApacheCamel | 609794 | 285996 | 169224 | 73833 | 80741 | 128720.0 | 4676 | 27.527802 | 27% | 45% | 740358 | 379198 | 181299 | 62703 | 117158 | 8871 | 24% | 54% |
| ApacheContinuum | 104122 | 44893 | 18675 | 23854 | 16700 | 22972.0 | 493 | 46.596348 | 17% | 75% | 34610 | 18567 | 4517 | 5370 | 6156 | 175 | 13% | 24% |
| ApacheCxf | 519661 | 282322 | 102747 | 68265 | 66327 | 122579.0 | 3587 | 34.173126 | 19% | 56% | 396843 | 234402 | 69968 | 31886 | 60587 | 3018 | 17% | 43% |
| ApacheDerby | 719210 | 317023 | 218875 | 85086 | 98226 | 124954.0 | 1915 | 65.25013 | 30% | 58% | 514042 | 304912 | 96713 | 39811 | 72606 | 1021 | 18% | 41% |
| BouncyCastleJAVA | 506955 | 237742 | 84105 | 117879 | 67229 | 98778.0 | 2992 | 33.014038 | 16% | 72% | 191818 | 119967 | 10026 | 28631 | 33194 | 668 | 5% | 27% |
| Cassandra | 370056 | 186028 | 60852 | 75586 | 47590 | 100552.0 | 1536 | 65.46354 | 16% | 75% | 122340 | 73484 | 12951 | 17190 | 18715 | 429 | 10% | 24% |
| Eclipse BIRT | 1971169 | 861414 | 502940 | 373758 | 233057 | 344724.0 | 8166 | 42.21455 | 25% | 84% | 371660 | 193812 | 78700 | 42673 | 56475 | 1607 | 21% | 15% |
| EclipseLink | 1280593 | 556273 | 417527 | 147860 | 158933 | 223206.0 | 5851 | 38.14835 | 32% | 47% | 1409466 | 798265 | 266439 | 134940 | 209822 | 11397 | 18% | 52% |
| Elasticsearch | 456475 | 230626 | 110991 | 56207 | 58651 | 102396.0 | 3130 | 32.71438 | 24% | 61% | 286569 | 190812 | 34097 | 23690 | 37970 | 1275 | 11% | 38% |
| Errai | 193159 | 86075 | 59033 | 20835 | 27216 | 33898.0 | 1939 | 17.482208 | 30% | 66% | 99234 | 54703 | 16873 | 9880 | 17778 | 1427 | 17% | 33% |
| Freedomotic | 84864 | 39993 | 26999 | 8189 | 9683 | 13239.0 | 526 | 25.1692 | 31% | 93% | 5614 | 2590 | 2005 | 413 | 606 | 47 | 35% | 6% |
| GeotoolKit | 1540034 | 594354 | 660405 | 133245 | 152030 | 248680.0 | 7850 | 31.678982 | 42% | 89% | 181545 | 100487 | 46868 | 9670 | 24520 | 1007 | 25% | 10% |

*Table 4.1: Sample of the outcomes of our dataset analysis*

## 4.2 Statistics

During the conducted study, some statistics and numbers were calculated and extracted from our numerical data, those statistics were used later on to answer our research questions. The main numbers of those statistics are tabulated in the following table:

| | TEST CODE % |
|---|---|
| **MIN** | 0.00% |
| **MAX** | 63.00% |
| **AVERAGE** | 28.51% |
| **MEDIAN** | 29.00% |
| **STANDARD DEVIATION** | 15.26% |

*Table 4.2: Statistics founded during the study*

However, the numbers mentioned above do not explain much. Table 4.3 details the prevalence of the percentage of the test code written among our dataset and shows their distributions; only 1.25% of the projects have more than 60% of their total code as test code, 6.25% of the projects have between 50-59% test code, 22.5% have 40-49%, 20% have 30-39% and the rest of the projects have less than 30% of their code as test code.

| PROJECTS % | # OF PROJECTS | TEST CODE % |
|---|---|---|
| **1.25%** | 1 | 60-69% |
| **6.25%** | 5 | 50-59% |
| **22.5%** | 18 | 40-49% |
| **20%** | 16 | 30-39% |
| **17.5%** | 14 | 20-29% |
| **18.75%** | 15 | 10-19% |
| **13.75%** | 11 | 0-9% |

*Table 4.3: Prevalence of the % of Test Code in Our Dataset*

The following figure visualizes the numbers mentioned in the previous table:

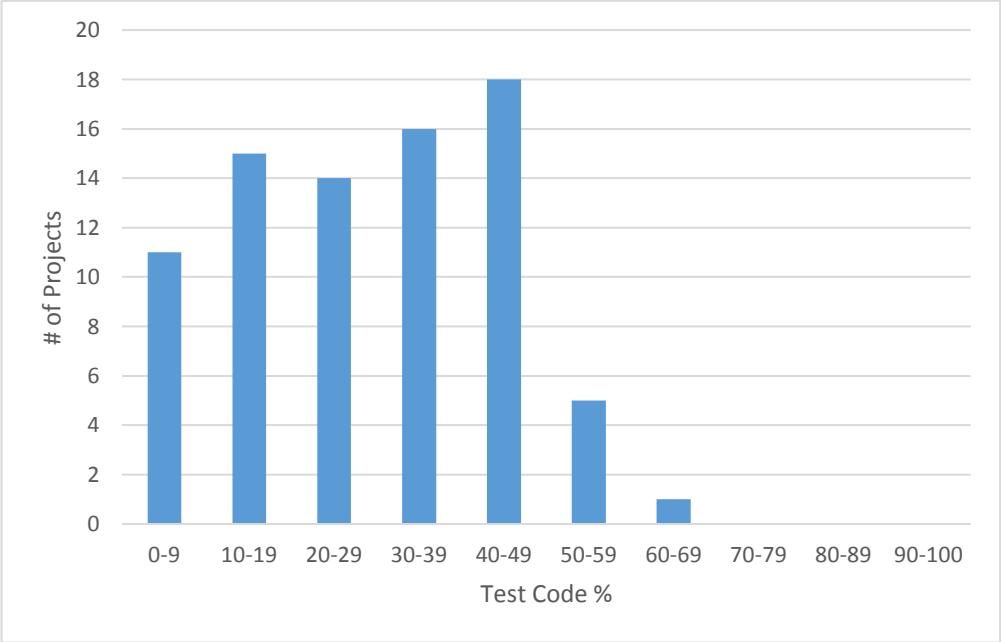


*Figure 4.1: Prevalence of the % of Test Code in Our Dataset*

## 4.3 Developers and Test %

Developers mean a lot in software development process as they participate by writing code, developing test cases, and solving errors. Thus, finding a correlation between the number of developers and written test code is necessary, to see whether there is any impact of those developers on the amount of test code done in those projects. Our dataset consists of 80 real world projects with a total of 54,183,278 lines of code. According to our statistics, those projects were written by 11,206 developers. Figure 4.2 shows the correlation, if any, between the developers and test code written in the projects in our dataset.

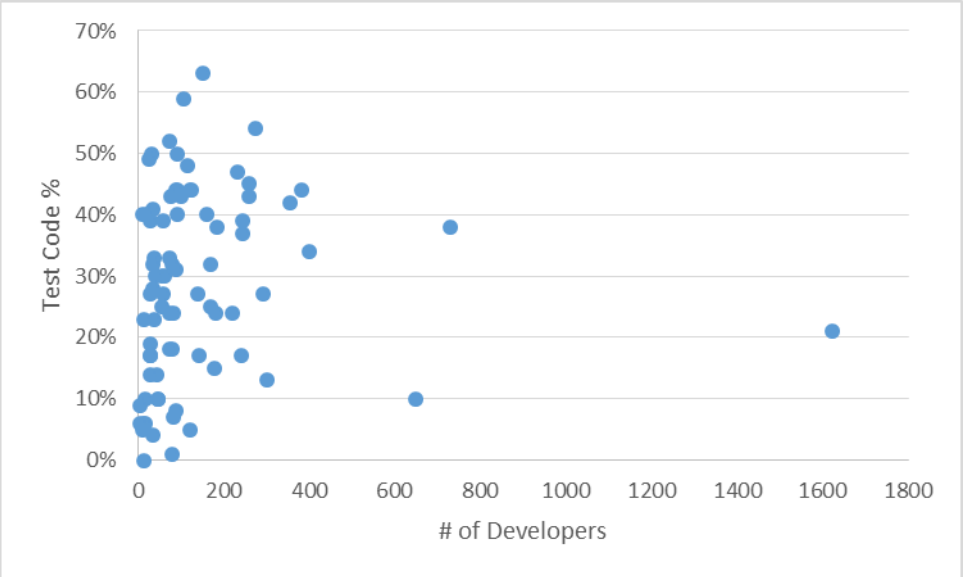


*Figure 4.2: Correlation between # of Developers and Test Code %*

## 4.4 Projects Size and Test %

The project size in a matter of Line of Code amount defines the amount of work done in the software development process. As the relation between project size and the test code percentage could be important for field workers, we have tried to find whether if there is any correlation between projects size and test code that was written in those projects. Figure 4.3 illustrates the correlation, if any, between those two factors.

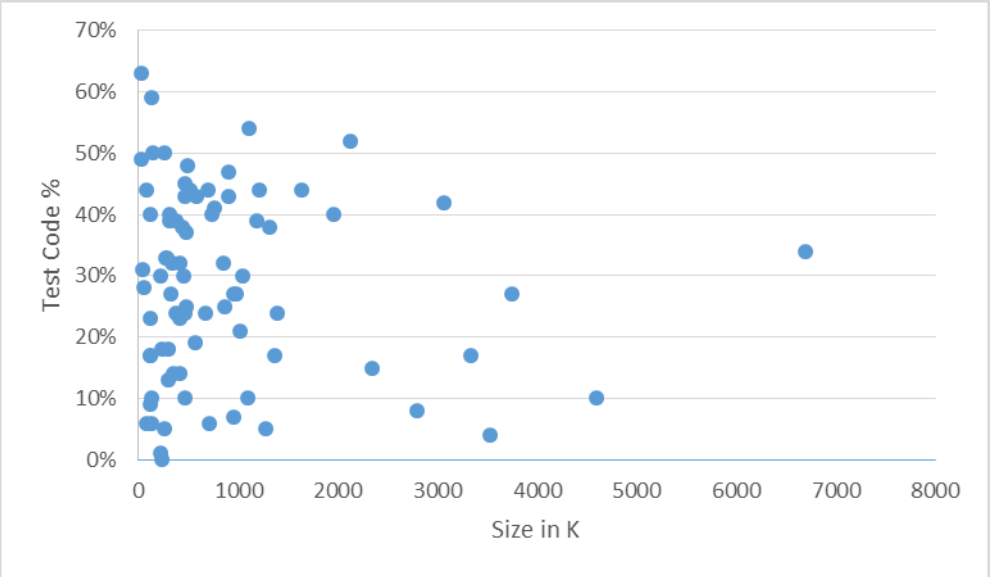


*Figure 4.3: Correlation between Project Size and Test Code %*

# 5 Analysis

As far as we know, this is the first study which investigates the relation between the amounts of test code written with different attributes of the project, using real world projects. According to the results above, we do not see any strong relation or distribution that the projects follow when it comes to testing, or in other words, they have a normal distribution. However, to ensure that we make some statistical measurements.

## 5.1 Statistical Measurements

Although that the figures of the correlations (Figure 4.3, 4.2) do not show a clear correlation, neither between the percentage of test code and number of developers nor the size of the project. However, we make some statistical measurements to ensure that. We calculate the Spearman's rho. Spearman's rho ($\rho$), also called Spearman's rank correlation coefficient, is a nonparametric measure used to assess the relation between two variables (x,y) statistically[10]. The following equation shows the formula that is used to find the coefficient:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

Where:
$d_i$ = the difference between each rank of corresponding values of x, y.
$n$ = the number of pairs of values.

After applying Spearman's rho on the test code percentage and 1. Projects size 2. Number of contributors we found the following results: 0.212470699 0.348922 respectively. However, the values of ρ has an interval between [-1;1] where 1 indicates a strong correlation, -1 indicates a strong negative correlation and 0 indicates no correlation. By looking at the $\rho$ we got, we realize that there is a little if any (linear) correlation between the percentage of the test code written and project size, as well as a weak correlation between test code percentage and number of developers, which is compatible with what has been found in previous studies.

## 5.2 Categorizations and Recommendations

From the outcomes of the conducted research, the results and the statistical measurements, we may be able to say the following:

- The average percentage of test code found was 28.51% or ≈ 30%. Thus having less than this percentage means that you are below average, and that you may need do some more tests, to reach the average level.
- Having a percentage of a test code between 30-39%, means that you have enough testing in your project to be comparable with a program like: Elastic Search.
- 40-49% of test code, means that your project is comparable with a program like: Apache ActiveMQ when it comes to testing.
- 50-59% then your project is comparable with: Apache Camel.
- 60-69% at this level of testing, you would be reaching a really high level of test, and your program can be compared with Junit 4, which is a testing framework by itself.

# 6 Discussion

In this research, we have three research questions. That we are aiming to find a satisfying answers for. Our analysis lead us to the following answers:

RQ1. Do real world projects vary with respect to the percentage of test code written for a project? If yes, which distributions do they follow?
Yes they do. We study 80 real world projects with sizes from 30K LOC up to 6.7M LOC. We find a large distribution of test code percentage from 0% to 63%. We try also to find if they follow specific distributions. However, According to our findings, we find that they approximately follow a normal distribution.

RQ2. Which classifiers can be used to distinguish the distributions? We test the number of developers and project size as classifiers having an impact on the test percentage. Although we find a correlation of 0.34 but it's still weak. Thus, it cannot be used to distinguish the distributions based on the number of developers or project size.

RQ3. Are they thresholds/ranges for projects of a certain classification which can be used for recommendations?
We find that 28.5% of test code is considered as average, while percentages like 60% are hard to find. Thus, according to our results, a project with 30% of test code will be considered as a reasonably good tested project as it's above average, and in that case, the project can be compared with a project like OpenJDK, in matters of testing.

# 7 Conclusion and Future Work

We conduct a study over 80 real world projects to find a relation between test code percentage and a quality of a software. We plot graphs to show the distributions between test code percentages among the projects in our dataset. We also find correlations between the size and the developers of the projects and the amount of test code. We use the results to answer the research questions and to give recommendations. However, this study is conducted on open source projects, thus we do not know if the answers and the recommendations are valid in the case of closed source projects.

In this research, we have limited our study to 80 open source projects and to some characterizes of the projects. In the future, we intend to expand our studies, and conduct it on a larger scale to include more projects. We can also consider studying closed source projects if possible, to see if the results may differ. Further, we also plan to examine automated tested projects and calculate the test coverage in those projects.

# 8 References


[1] K. Okumoto and L. A. Goel, "When to stop testing and start using software?," *ACM SIGMETRICS Performance Evaluation Review*, pp. 131-138, 1981.

[2] P. J. L. LIONS, "Flight 501 Failure," European Space Research Institute, Paris, 1996.

[3] B. Beizer, Software testing techniques (2nd ed.), New York, NY, USA: Van Nostrand Reinhold Co., 1990.

[4] P. S. KOCHHAR, T. F. Bissyande, L. David and L. JIANG, "Adoption of Software Testing in Open Source Projects: A Preliminary Study on 50,000 Projects," in *School of Information Systems at Institutional Knowledge*, Singapore, 2013.

[5] "The Black Duck Open Hub," Black Duck Software, Inc., [Online]. Available: https://www.openhub.net/. [Accessed 27 05 2016].

[6] "GitHub," Wikimedia Foundation, Inc, [Online]. Available: https://en.wikipedia.org/wiki/GitHub. [Accessed 27 05 2016].

[7] "Cyclomatic Complexity," Wikimedia Foundation, Inc, [Online]. Available: https://en.wikipedia.org/wiki/Cyclomatic_complexity. [Accessed 01 06 2016].

[8] Research Triangle Institute, "The Economic Impacts of Inadequate Infrastructure for Software Testing," NIST Publications, 2002.

[9] Stecklein, Jonette M., Dabney, Jim, Dick, Brandon, Haskins, Bill, Lovell, Randy and Moroney, Gregory, "Error Cost Escalation Through the Project Life Cycle," in *NASA Johnson Space Center*, Houston, TX, United States, 2004.

[10] "Spearman's rank correlation coefficient," Wikimedia Foundation, Inc., [Online]. Available: https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient. [Accessed 10 09 2016].

[11] A. Bansal, "Assessing the Relation Between Quantity of Test Cases and Amount of Code Coverage," *International Journal of Engineering and Advance Technology*, vol. 3, no. 5, pp. 184-188, 2014.

[12] P. S. Kochhar, T. F. Bissyande, D. Lo and L. Jiang, "An Empirical Study of Adoption of Software Testing in Open Source Projects," in *13th International Conference on Quality Software*, Najing, China, 2013.

[13] M. C.C., K. v. Wyk and W. Radosevich, "Risk-Based and Functional Security Testing," US-CERT United States Computer Emergency Readiness Team, 27 09 2005. [Online]. Available: https://www.us-cert.gov/bsi/articles/best-practices/security-testing/risk-based-and-functional-security-testing#refs. [Accessed 22 12 2016].

# 9 Appendix 1

A List over the analyzed projects name in our study:

| | | |
|---|---|---|
| Activiti | Apache Syncope | Junit 4 |
| Apache ActiveMQ | Apache TomEE | Lens Kit |
| Apache Ant | AWS SDKJava | Liferay Portal |
| Apache Apex | Bio Formats | Midpoint |
| Apache Archiva | Bouncy Castle JAVA | Mockito |
| Apache Axiom | Byte Buddy | ModeShape |
| Apache Camel | Cassandra | Mule Community |
| Apache Commons Lang | Cayenne | My Tourbook |
| Apache Continuum | Checkstyle | Neo4j |
| Apache Cxf | Commons IO | Nifty-gui |
| Apache Derby | Data Cleaner | Nutch |
| Apache Directory Studio | Eclipse BIRT | Nuxeo |
| Apache Ehcache | Eclipse Link | Onos |
| Apache Flink | Elasticsearch | Open Cms |
| Apache FOP | Errai | Open DJ |
| Apache Hadoop | Freedomotic | Open JDK |
| Apache HBase | Geotool Kit | Open MRS Core |
| Apache Hive | Geo Tools | Rapid Miner |
| Apache Ignite | H2 Database | Seed Stack |
| Apache ISIS | Hazelcast | Spring Framework |
| Apache Jackrabbit Oak | Infinispan | Stream Sets |
| Apache Jclouds | JBoss Drools | Switch Yard |
| Apache Nutch | Jena | Weka |
| Apache PDF Box | Jenkins | Wicket |
| Apache Pig | Jersey | Wild Fly |
| Apache Pivot | Jetty | Ya Cy |
| Apache Storm | J Rebirth | |