Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

Test Script and Application Code Relative Complexities

1 INTRODUCTION

- Scope of the project:

  The scope of the project is to basically compare the complexities (Line of Code and Cyclomatic particularly) of the test as well as the application code. But it involves the separation of application and test code as its first step. Using different tools and programs the separation of code and then their complexity measurement will take place. The project study will be done on the complexity metrics of the test and application code availed from twelve different open source software. The detailed description of the steps involved is given under 'Design of the Case Study' [3] section of the report. The study mainly revolves around the cyclomatic complexity and line of code complexity and there are few things which are out of the scope of this project and mentioned inline as limitations under section [2.2].

- Goal of the project:

  The goal of this project is to study the relative complexities of test code and application code. And based on the analysis it needs to be figured out which one possesses relatively more complexity – test code or application code. Can the test script created to test the application become more complex than the application code itself? Their different level of complexities will be studied in detail. Different statistics and graphs would be used while investigating the relative complexity.

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

- Structure of the report:

Contents

2 STATE OF THE ART OVERVIEW

2.1 Brief Overview of the Theory:

Measuring the internal quality of source code is one of the traditional goals of making software development into an engineering discipline. Cyclomatic Complexity (CC) is an often-used source code quality metric, next to Source Lines of Code [1]. Before discussing the complexities of test code and application code, it would be better to understand about the complexity metrics that will be used in this project:

Line of Code (LOC): *A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements* [2].

Cyclomatic Complexity (CC): *The cyclomatic complexity of a program is the maximum number of linearly independent circuits in the control flow graph of the said program, where each exit point is connected with an additional edge to the entry point* [3].

According to McCabe [3], the CC number can be computed by counting forks in a control flow graph and adding 1, or equivalently counting the number of language constructs used in the Abstract Syntax Tree (AST) which generate forks ("if", "while", etc.) and adding 1. This method is easy, but it depends on the programming language used for the source code.

The analysis is done based on the statistics, and so, few basic terminologies of statistics which have been used in this report are defined below:

Histogram: *A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable). It differs from a bar graph, in the sense that a bar graph relates two variables, but a histogram relates only one.* [4]

Descriptive Statistics: *Descriptive statistics are broken down into two categories. Measures of central tendency and measures of variability (spread). Central tendency refers to the idea that there is one number that best summarizes the entire set of measurements, a number that is in some way "central" to the set. For instance, median, mean, etc. The measure of spread* *refers to the idea of variability within your data. Like, standard deviation, quartile, skewness.* [5]

Mean: *Mean or Average is a central tendency of the data i.e. a number around which a whole data is spread out. In a way, it is a single number which can estimate the value of the whole data set.* [5]

Median: *It is the value which divides the data into two equal parts i.e. number of terms on the right side of it is same as the number of terms on the left side of it when data is arranged in either* ***ascending or descending order****.* [5]

Standard Deviation: *It is the measurement of an average distance between each quantity and Mean. That is, how data is spread out from Mean. A low standard deviation indicates that the data points tend to be close to the mean of the data set, while a high standard deviation indicates that the data points are spread out over a wider range of values.* [5]

Quartiles: *In statistics and probability, quartiles are values that divide your data into quarters provided data is sorted in an* ***ascending order.*** *There are three quartile values. First quartile value is at 25 percentile. Second quartile is 50 percentile, and Third quartile is 75 percentile. Second quartile (Q2) is the median of the whole data. First quartile (Q1) is median of upper half of the data. And Third Quartile (Q3) is median of lower half of the data.* [5]

2.2 High level Details of Methodologies and Techniques used:

Having a brief idea about the theory of complexities, moving further to the actual methods and techniques that are used in the report. The short description of the steps involved, and the tools required are as follows, the detailed discussion has been done in the next section [3].

First, the test code and the application code are to be separated from the given code for the different case studies. To accomplish this step, a couple of tools would be used apart from the manual segregation approach. After separating the codes, complexities would be measured for both the codes using Understand tool, as described in the proposal of the project. Based on the available data of the complexities generated by the tool, different graphs would be plotted for the proper analysis to conclude the relative complexities of the test and the application code. The tools which are needed are listed in Table 2.1.

S.No.	Tool/Software	Use
1	Understand 5.0.971 *	To generate the metrics of the codes
2	Python 3.7 *	Few scripts are written to separate test code from application code, and to generate the different plots based on the metrics data available

Table 2.1 Tools needed to be installed

Though it has been tried to cover the maximum analysis possible, considering the limited and time-constrained duration of the project, few things are still out of scope for now, which can be extended later. These are termed as the limitations for this project:

Limitation 1: The cyclomatic data of only different functions or methods# (Function, Public Function, Private Function, Method, etc.) is used for analysis. The data having zero or null values for Cyclomatic Complexities in the Metrics files generated by Understand tool are ignored. Because the focus is only on the Cyclomatic data which occurs at the function level. To study the cyclomatic complexity at the class or file level, SumCyclomatic is to be considered for further study.

Limitation 2: The cyclomatic complexities are not studied at the specific programming languages level. That means the code is mixed of files of different programming language. The comparison of complexities of different programming languages internally was quite difficult to include into the scope of the project.

** The steps to download these tools can be found online. Kindly refer ReadMe.txt for details.*

# The test and application code having Java as its programming language contains 'Method' as its Kind, and not 'Function', in the Metrics generated by Understand tool.

3 DESIGN OF THE CASE STUDY

To discuss the design of the case study, first of all, the entire project needs to be divided into the below major sections:

3.1 Collection of Open Source Project:

The study of the relative complexity is done based on the code available from twelve different open source projects, the list with the URL is mentioned in Table 3.1.

S.No.	Open Source Project	URL
1	Apache ant	https://github.com/apache
2	Curl	https://github.com/curl/curl
3	Jkarta JMeter	https://github.com/apache/jmeter
4	JRuby	https://github.com/jruby/jruby
5	Voldemort	https://github.com/voldemort/voldemort
6	Abiword	https://github.com/AbiWord/abiword
7	Client	https://github.com/owncloud/client
8	Foundation DB	https://github.com/apple/foundationdb
9	Keepass	https://github.com/keepassx/keepassx
10	Muse Score	https://github.com/musescore/MuseScore
11	Scintilla	https://github.com/notepad-plus-plus/notepad-plus-plus/tree/master/scintilla
12	Power Shell	https://github.com/PowerShell/PowerShell

Table 3.1 List of Open Source Projects

These open source projects are downloaded from the given GitHub links and stored in folder.

3.2 Separation of test code from application code:

It is rightly mentioned in the proposal that it is quite difficult to precisely distinguish test code from application code: in some open source projects, test code is mixed with application code, which hinders precise measurement and introduces threats to validity. But after having multiple discussion, eventually, it is concluded that there is no exact way to decide the basis on which precise segregation is possible. However, it is possible to achieve the maximum accuracy by finding the best way possible. To find the test code out of application code, the two approaches are followed:

(I) Finding Test folders and files by name: The basic and most popular approach is to find the files or/and sub-folders which are inside the folder having the name/s 'Test/test/TEST'*. And the other way is to keep looking for the same name for files too which are not inside a folder having 'Test/test/TEST' as its name. To achieve this, two python scripts are written: **TestCodeFilter.py**. TestCodeFilter.py asks for the path of the folder, where all the open source projects are stored, as an input. Once the input is given to this script, it will traverse all the folders and files for each open source project. It will keep storing the directory path of each and every folder/file in a.txt file. Now in the next step, from this .txt file, it will search for 'Test/test/TEST' substring in the directory path line by line. Because each line in that .txt file will have this substring if it has any folder or file name having 'Test/test/TEST'. For an example, the path- D:\..\SYSC 5105 Software Quality\Project\SQ Codes\abiword-trunk\abiword-trunk\**test**\wp has a folder having name 'test' will be filtered. All such filtered paths will be stored into a new tfile.txt file which apparently has all the test files. The remaining paths will be stored in a different afile.txt file.

(II) This is not accurate to assume that only files/folders with 'Test' name will have the test code. Thus, apart from this automatic separation of the test code from application code, documentation of each open source project is also considered if it has any information about the test files/folders. For an instance, in the documentation (/docs/Testing.txt) Abiword open source project, it is mentioned that:

"Test are __.t.cpp files. Usually in the t subdirectory of what they are testing."

As the instructions are given by the documentation, the paths of other test files are also moved into tfile.txt from afile.txt created in the step (I).

Now, two files are there: tfile.txt having paths of probably all the test codes and afile.txt having the remaining i.e. application codes.

** It has been manually checked that no open source project contains Test written in any other format like 'TEST', etc. That's why checking only the four formats of string Test as mentioned.*

Using another python script **FileCopy.py**, it will automatically keep creating two different folders (one for test code and second for application code) under each open source project folder. This way we have finally segregated test code from the application code and stored those into two different folders.

3.3 Generating Complexity Metrics

Once the test code and application code are separated for each open source project. Next step is to generate the complexity metrics of these codes separately for each case study. The use of Understand tool is explained step by step in details in the Using-SciTools-Understand.pdf file which is available with this report package. These steps are taken for every test and application code folders, of each open source project, which were created as mentioned at the end of section 3.2. There are twenty-four folders in total, two (test and application) for each case study. In the Understand tool, every time after creating a new project, one of these twenty-four folders are selected to generate the metrics at a time. While generating the Detailed Metrics, only CountLineCode (for LOC), Cyclomatic (for CC), and SumCyclomatic (CC at the file level) are checked from the list of available metrics because only these are relevant as per the scope of this project. And by doing so, twenty .csv files with the LOC and CC data of various kinds (Function, File, Class, etc.) are available to use. And all the .csv files generated by the Understand tool are stored in a separate folder. Now by using the python script of **csvFilter.py**, the path of these .csv files are stored in a new .txt file which would be used later while plotting all the relevant graphs using some other python script in the next step. Before generating the Detailed Metrics at each step, the information about the programming languages is also stored in a file 'LanguageOfCode.doc' which can be referred to in the package of the project.

3.4 Plotting different graphs needed for the analysis of complexity data

To achieve the main goal of the project which is eventually to study the complexity metrics gathered in the previous step, different graphs are needed to be plotted. For creating these graphs, the python script used is - **GraphPlotter.py**. Before using this script, based on the programming language (C or Java), the .txt file, which was just created in the previous step which has the paths of all .csv files, is divided into three different .txt files- one with no Java language, one with only Java language, and the remaining with all the languages. The reasoning is explained in the appendix section (1) in details. The GraphPlotter.py basically traverses these .txt files and for each .csv file, it plots four different graphs and keeps on saving these files automatically under the path which it has asked as an input. The different graphs are mentioned in Table 3.2. The first two graphs are plotted only for the data related to the functions, logic is written in GraphPlotter.py to filter only those data whose Kind is Function. Whereas the last two are plotted based on the data at the file level.

S.No.	**Different Graphs plotted by GraphPlotter.py**
1	Line graph between Cyclomatic Complexity and Functions of that test/application code
2	Line graph between Line of Code and Functions of that test/application code
3	Line graph between SumCyclomatic Complexity and Files of that test/application code
4	Line graph between LOC and Files of that test/application code

Table 3.2 Different graphs generated from GraphPlotter.py

Though these plots show the detailed distribution of the complexities for each and every function and file present in the test and application code, it is quite difficult to study the relative complexities based on these graphs. Moving forward, another python script **HistoPlotter.py** is written to generate the histogram graphs for all the CC and LOC data of different functions or methods. It also creates the Statistics Description table for both LOC and CC data of each metrics file as mentioned in Table 3.3. And it is processed to keep on saving these graphs and tables as an image file to the destination folder. Based on these plots (2), the relative complexities of the test and application code can be observed.

S.No.	**Graphs and Tables to be analyzed**
1	Histogram graph of Cyclomatic Complexity for all functions of that test/application code
2	Histogram graph of LOC for all functions of that test/application code
3	Descriptive Statistics table of Cyclomatic Complexity for all functions of that test/application code
4	Descriptive Statistics table of LOC for all functions of that test/application code

Table 3.3 Graphs and Tables created from HistoPlotter.py

4 ANALYSIS OF THE RESULTS

As a result, a significant amount of data has been gathered for the analysis. From the previous section [3.4], numerous meaningful graphs presenting a different set of data are collected, however, the focus is to analyze the cyclomatic complexity and the line of code complexity of test code as well as application code. Thus, the main purpose of this project would be to analyze the data gathered as mentioned in Table 3.3. Because it is easier to study and therefore analyze the data of test and application code based on the descriptive statistics and distribution graphs of their complexities. Data to be analyzed is filtered at the function level (method level in case of Java language) i.e. all the non-functional data like File, Class, etc. are not considered because they give either 0 or null values for the Cyclomatic complexity (3).

The complexity can be compared by the various statistical aspects like mean, median, distribution (standard deviation) but in my point of view, the third quartile is a very important factor in deciding the complexity. A code can be considered complex by the highest cyclomatic values of the data. By its definition, the third quartile data is the last 25% section when the data is in ascending order. So, the last 25% contains the highest Complexity values or functions with maximum Complexity values. That means, whichever code contains more complex functions can also be considered possessing relatively more complex code. Thus, the third quartile (75%) is one of the major deciding factors when it comes to analyzing the relative complexity. A high value of the third quartile would be considered as having functions of higher complexity.

Even from the testing point of view, going by the definition of Cyclomatic Complexity; it depends on the number of linearly independent paths. More test cases are to be considered for more complex functions i.e. functions having higher CC values.

4.1 Case Study 1 [Apache Ant]:

4.1.1 Application Code Analysis:

The very first case study of Apache Ant is based on more than ten thousand functions of the application code. Fig 4.1.1 shows the distribution of cyclomatic complexity data of the application code at the function level. And the graph can be seen showing positive skewness as histogram's tail has a positive skew to the right. And which means a huge number of functions are of very less complexity value i.e. majority of the functions are smaller functions. It can be seen in the Fig 4.1.1 that close to seven thousand functions have the cyclomatic value of less than 2. The median of the data or the second quartile (50%) is 1. It is easily visualized that the few of the higher values of CC are outliers in the data because they are few in the numbers as compared to most of the data. For an instance, the max value of CC is 49.0 as mentioned in Table 4.1.1, which is an outlier in this case.

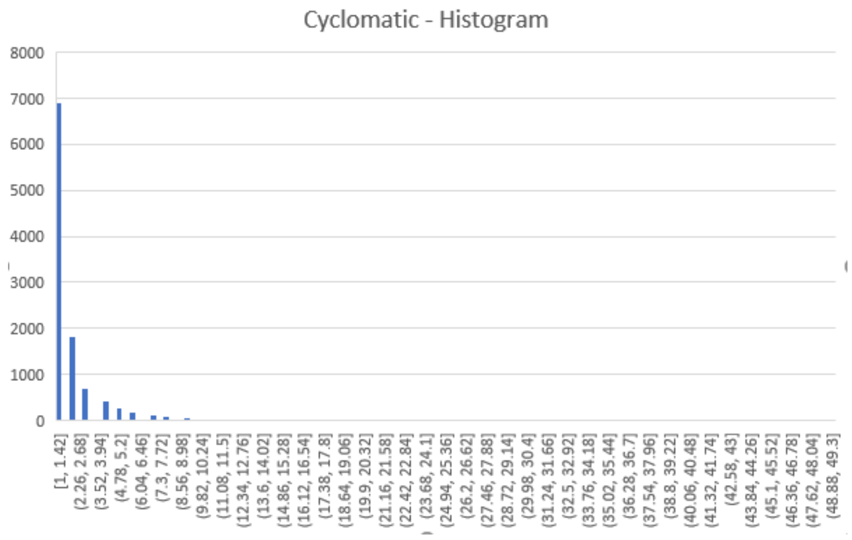
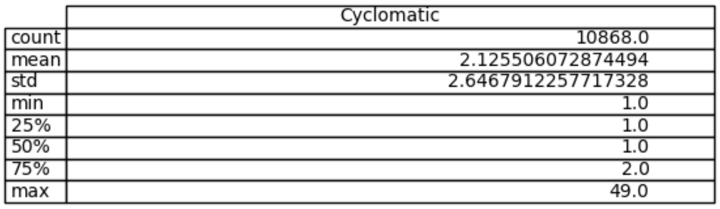


Fig 4.1.1 Distribution of CC for different Functions of Application Code (Apache Ant)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca



	Cyclomatic
count	10868.0
mean	2.125506072874494
std	2.6467912257717328
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	49.0

Table 4.1.1 Descriptive Statistics of CC for different Functions of Application Code (Apache Ant)

Out of these 10868 functions, almost 65% are of very small size i.e. less than 10 LOC and even the value of third quartile (75%) is 8 as mentioned in the descriptive statistics table of LOC of the application Table 4.1.2. This value is also near about similar to the mean value of the data. In this data of the application code, it has the maximum size of the file with 169 lines of code.

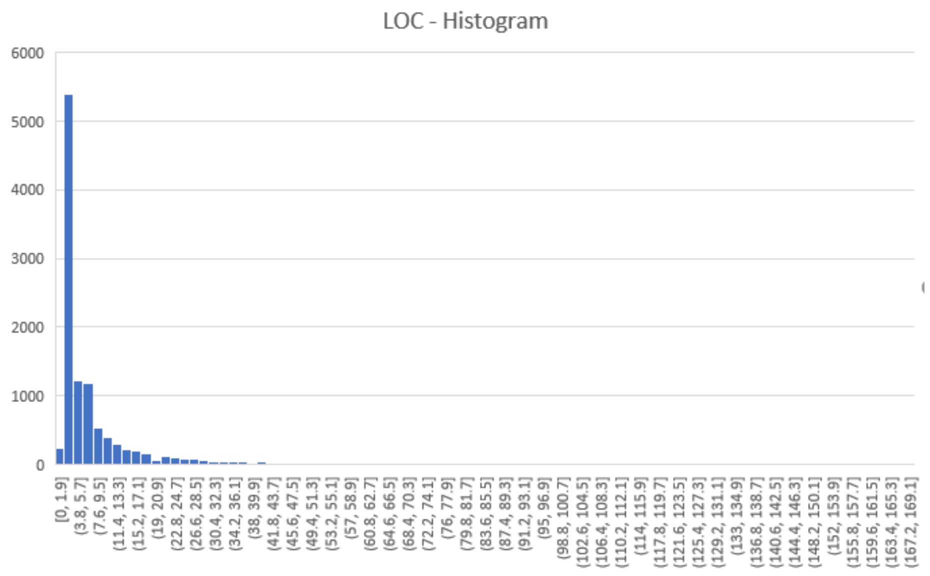


Fig 4.1.2 Distribution of LOC for different Functions of Application Code (Apache Ant)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

	CountLineCode
count	10868.0
mean	8.254416635995584
std	12.27828131121931
min	0.0
25%	3.0
50%	3.0
75%	8.0
max	169.0

Table 4.1.2 Descriptive Statistics of LOC for different Functions of Application Code (Apache Ant)

It is quite observable that though both CC and LOC are possessing exponential distribution but there is no direct relation between these values. As an example, by comparing max values of Table 4.1.1 and 4.1.2, it can be seen that there is a difference of more than thrice between the CC and LOC values, and so both of these data reflect a different level of complexities.

4.1.2 Test Code Analysis:

Test code of Apache Ant contains less than four thousand functions and the number is not even half of what it had in its application code. Besides few hundreds of functions, almost all of these have the complexity value close to 1 as visualized in the Fig 4.1.3. It is also evident from the descriptive statistics table 4.1.3. The distribution of cyclomatic complexity is also highly right-skewed like the application code.

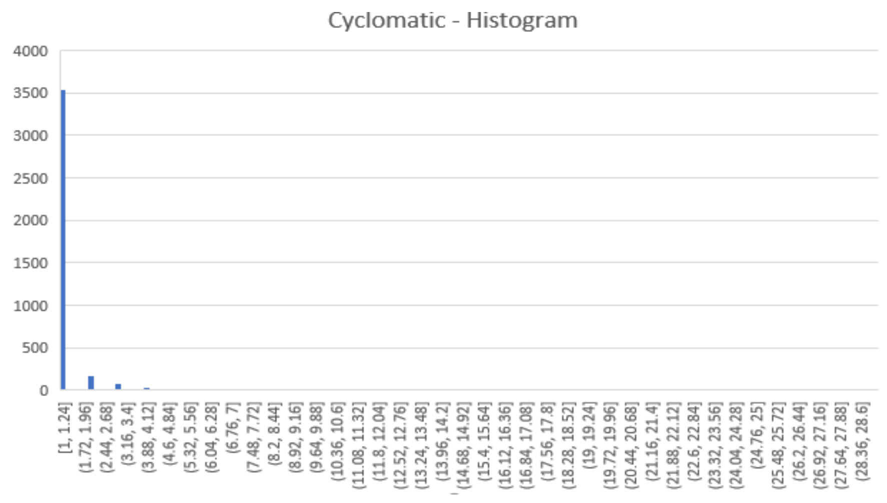
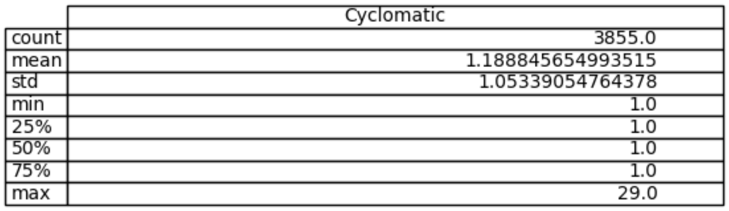


Fig 4.1.3 Distribution of CC for different Functions of Test Code (Apache Ant)



	Cyclomatic
count	3855.0
mean	1.188845654993515
std	1.05339054764378
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	29.0

Table 4.1.3 Descriptive Statistics of CC for different Functions of Test Code (Apache Ant)

The LOC plot of the test code (Fig 4.1.4) easily tells us that it has slightly more Complexity values as compared to the CC plot and the descriptive statistic figures are also in sync with this argument.

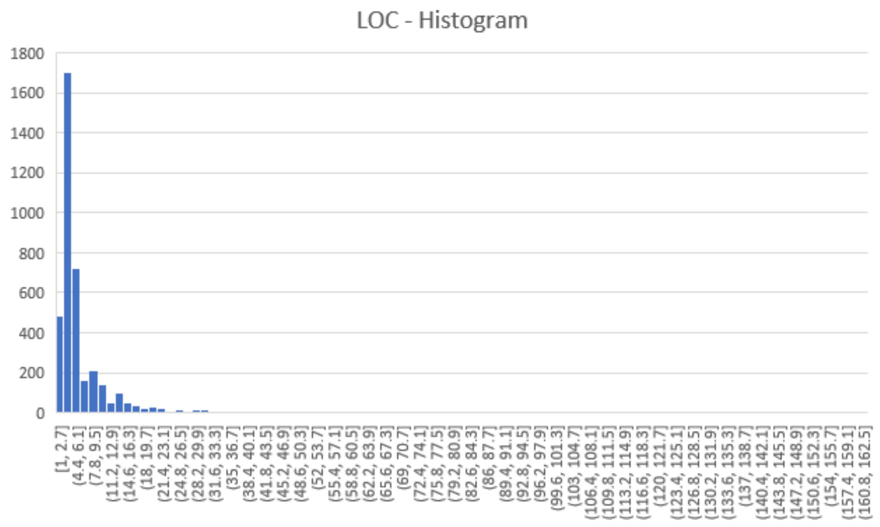
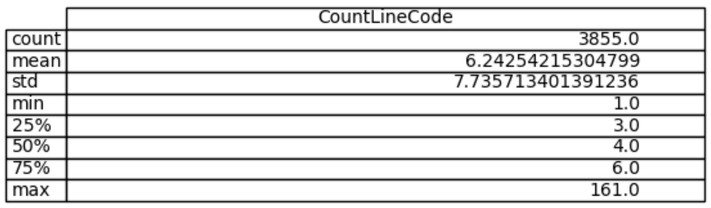


Fig 4.1.4 Distribution of LOC for different Functions of Test Code (Apache Ant)



	CountLineCode
count	3855.0
mean	6.24254215304799
std	7.735713401391236
min	1.0
25%	3.0
50%	4.0
75%	6.0
max	161.0

Table 4.1.4 Descriptive Statistics of LOC for different Functions of Test Code (Apache Ant)

Mostly test code functions comprise lesser lines of code, however, few outliers can be found as it was found in the previous case of application code data. For an example, the max value of LOC which is 161 is way higher than the LOC values of the remaining functions.

4.1.3 Relative Complexity of Test Code and Application Code

By comparing the graphs and tables of test code and application code, relative complexities can be observed. The CC data of application code has a standard deviation of 2.65 whereas the CC data of test code has a standard deviation of 1.05. The CC data of application code having a higher standard deviation tells that the application code CC data is more spread out. On the other hand, while comparing standard deviation of LOC data of the application and test code, it is observable that application code LOC data is more dispersed than the test code LOC data. Thus, it can be concluded that the complexity of application code has significantly higher distribution than that of test code.

By and large, the numbers are quite higher for the complexities of application code as compared to the test code – be it the number of functions, or max, min, mean of LOC/CC. Like, mean values of CC and LOC for application code is 2.12 and 8.25 respectively whereas for test code the values are 1.18 and 6.24. Third quartile (75%) of application code is higher than test code for both cases – CC and LOC.

Figures	**Application Code**	**Test Code**
Number of Functions	10868	3855
Third Quartile (CC)	2	1
Third Quartile (LOC)	8	6
Mean (CC)	2.12	1.18
Mean (LOC)	8.25	6.24
Median (CC)	1	1
Median (LOC)	3	4
Standard Deviation (CC)	2.65	1.05
Standard Deviation (LOC)	12.27	7.73
Max (CC)	49	29
Max (LOC)	169	161

Table 4.1.5 Statistics Comparison of Application and Test Code Complexities (Apache Ant)

It is clearly seen by observing all the figures that the bar chart of application code is significantly higher than that of test code for each comparison (in Fig 4.1.5) other than the slight conflict of the median value of LOC.

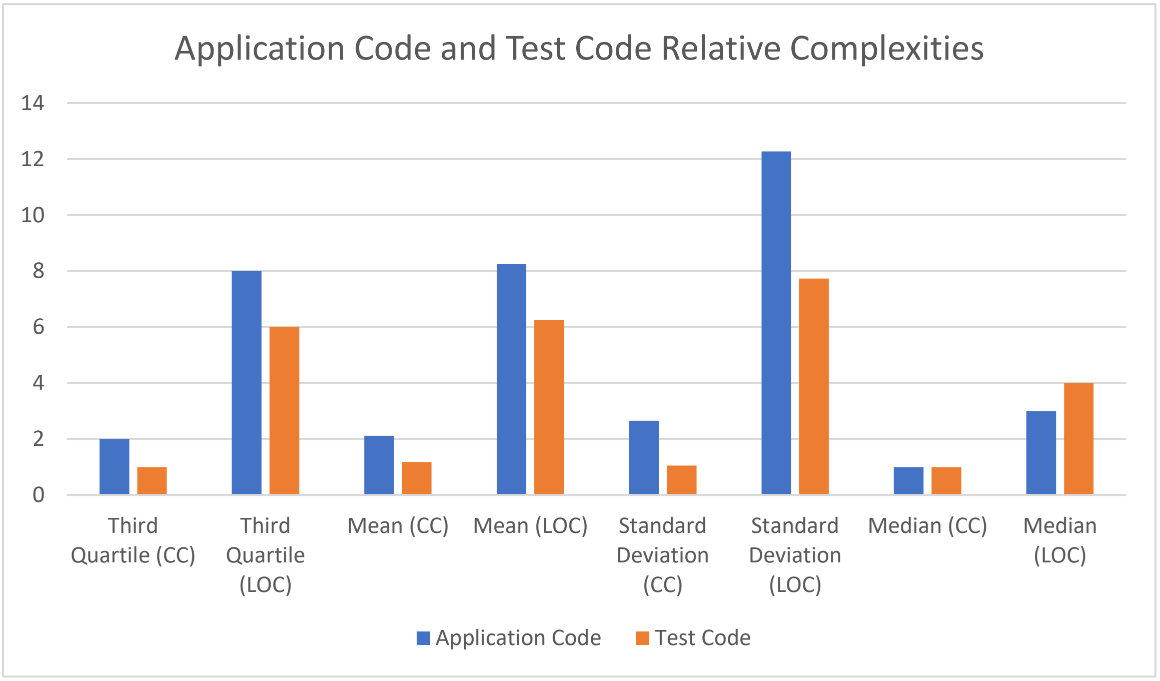


Fig 4.1.5 Comparative Statistical Analysis of Application and Test Code Complexities (Apache Ant)

Almost similar results are achieved in the case studies of Client (6), Curl (5), MuseScore (7), and Scintilla-Notepad (8) where nearly all the factors of application code are higher than that of test code for both cyclomatic as well as LOC complexities.

4.2 Case Study 2 [Foundation DB]:

The skewness and distribution of the histogram plots of application and test code of both cyclomatic and LOC are more or less similar to the previous case study of Apache Ant, however, the descriptive statistics is interestingly different, and which should be the focus of study in this case. Because discussing the histograms of this case study (4) would be a reiteration of what we have already discussed in the previous case study. Thus, only relative complexity would be the point of focus.

4.2.1 Relative Complexity of Test Code and Application Code:

The standard deviation data of application code is higher than that of test code in case of CC, so application code would have more dispersed complexity, whereas in the case of LOC; both of the codes show equal level of deviation and which is quite high and that means both have the highly deviated data as compared to the respective mean values. Cyclomatic descriptive figures of application code are slightly higher when compared to the test code. However, in case of LOC, the descriptive figures of test code (third quartile = 13, mean = 15.72, median = 6) are quite higher than the values of application code (third quartile = 11, mean = 12.59, median = 4) as mentioned in Table 4.2.1. Whereas, the max values, which is also the complexity value of function having maximum complexity in terms of cyclomatic and line of code for CC and LOC respectively, is extremely high for application code when compared with test code.

Figures	**Application Code**	**Test Code**
Number of Functions	14728	1149
Third Quartile (CC)	3	3
Third Quartile (LOC)	11	13
Mean (CC)	3.01	2.61
Mean (LOC)	12.59	15.72
Median (CC)	1	1
Median (LOC)	4	6
Standard Deviation (CC)	8.74	5.56
Standard Deviation (LOC)	43.55	43.31
Max (CC)	610	79
Max (LOC)	3146	560

Table 4.2.1 Statistics Comparison of Application and Test Code Complexities (Foundation DB)

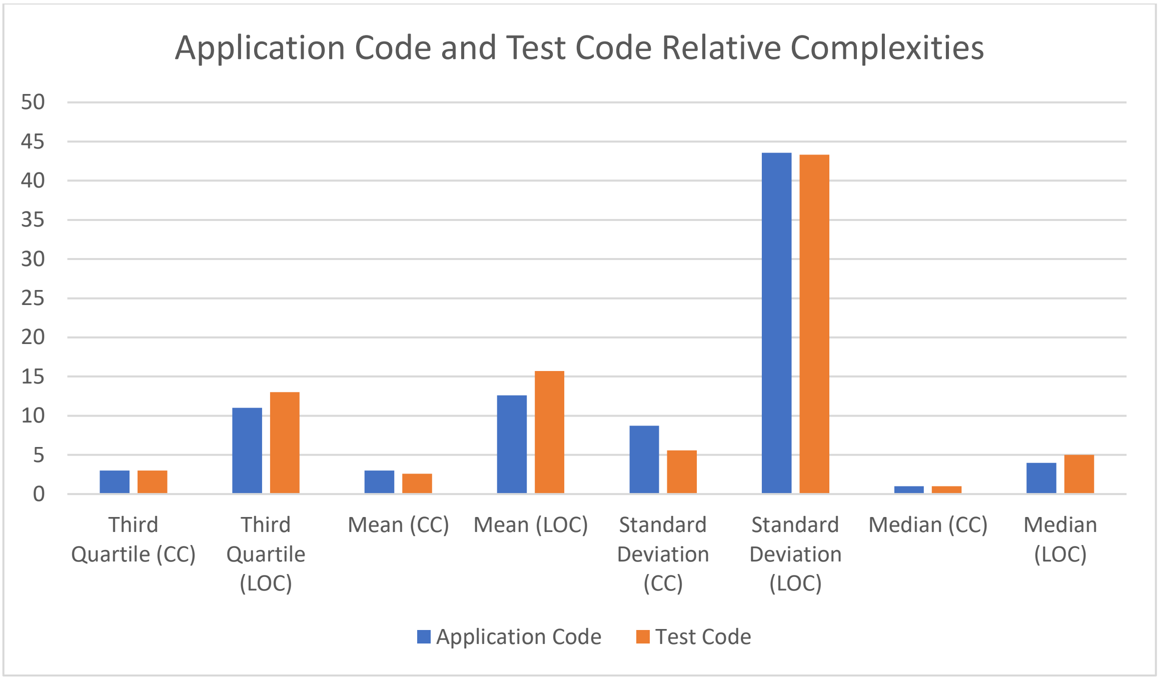


Fig 4.2.1 Comparative Statistical Analysis of Application and Test Code Complexities (Foundation DB)

Simple observation can be done by checking the bar charts plotted in Fig 4.2.1 that the cyclomatic complexity of application is more than that of test code and LOC complexity is more in test code as compared to the application code. However, it is just a relative comparison by ignoring the functions, but when the number of functions, which is almost twelve times more for application code than test code, and max values are compared, it can be considered that the overall complexity would still be higher for application code.

This consideration is applicable to all the case studies which have slightly higher Complexity values for test code. Other similar open source projects having higher cyclomatic complexity for application code and higher LOC complexity for test code are – Abiword (10), JMeter (11), Keepass (12), PowerShell (13), Voldemort (14). Whereas, JRuby (9) tends to show higher cyclomatic complexity for test code and higher LOC complexity for application code. But in all such cases, the strength of the number of functions always go in the favor of application code.

4.3 Summary of Important Results [Conclusion]:

Five out of twelve case studies show that the cyclomatic, as well as LOC, complexities are higher for application codes than test codes. In other seven cases, either cyclomatic or LOC complexity is comparatively higher for application codes. After considering the factor that application codes always contain a very huge number of functions, as compared to the test codes, which ultimately enhances the complexities which were slightly less when considering at the descriptive level. Thus, after deep analysis of an overall aspect of the data, it is evident to say that the application code is relatively more complex than the test script.

5 LESSONS LEARNED AND OPEN ISSUES

- The presumption was that an application code would always be more complex than its testing code. However, while studying the descriptive statistics of the data, in few of the cases, this was not the case (if the size of the code is not been considered).

- Comment about the linearity: It is not completely true that there is a linear relationship between CC and LOC, because there have been the cases when the LOC of the test code was more than that of application code even when the CC was not more, and vice-versa.

- Though this was my first experience of working with any testing tool, there have been no issues using Understand tool. The main difficulty at the beginning of the project was to identify the test code out of the open source package for each project. But that was well handled by the discussion and applying technical approaches like filtering based on the name by using a python script.

- Working on python with the use of matplotlib (Library for statistics) was also one of the new things to learn. Plotting graphs from a tool, and automatically saving it into a folder are interesting to learn more about it.

- Generating and analyzing the descriptive statistics is one of the remarkable things I got to know during this project.

- Overall, it had been an enriching learning of a wide range of things. Few things have been challenging like adjusting the bins to plot the histogram plots through python. If there were no time constraints, it could have been improved more. Also, the graphs and the data can also be compared using image processing. Thus, everything would have been automated.

- Open issues of this project are to study and analyze the complexity at the different programming levels and at the file level as well. Again, the project duration was the factor to restrict the study at function level only. Taking this research further, these issues can be discussed.

6 REFERENCES

[1] Landman, D.; Serebrenik, A.; Bouwers, E.; Vinju, J.J. Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions, 2016

[2] Conte SD, Dunsmore HE, Shen VY. Software Engineering Metrics and Models. Benjamin-Cummings Publishing Co., Inc.: Redwood City, CA, USA, 1986.

[3] McCabe TJ. A Complexity Measure. IEEE Transactions Software Engineering 1976; 2(4):308–320

[4] Pearson, K. "Contributions to the Mathematical Theory of Evolution. II. Skew Variation in Homogeneous Material". Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 1895.

[5] Sarang Narkhede, Understanding Descriptive Statistics, 2018 (https://towardsdatascience.com/understanding-descriptive-statistics-c9c2b0641291)

Appendix A Appendices

(1) In the metrics .csv file, in the column Kind; for C or python language, the term Function is used to refer the functions of the codes whereas, for the files of Java language, the term Method is used. So, it was hard for the python script to distinguish between function and method, that's why the .txt file containing the paths of .csv files is divided into three different .txt files based on the programming language and the below logic is implemented to create the dataframe:

```
If (file_list == 'csvFile_C.txt'):
   func_data = data[(data.Kind == 'Function')]
elif (file_list == 'csvFile_J.txt'):
   func_data = data[(data.Kind == 'Method')]
elif (file_list == 'csvFile_CJ.txt'):
   func_data = data[(data.Kind == 'Function')]
   func_data2 = data[(data.Kind == 'Method')]
   func_data.append(func_data2)
```

The 'LanguageOfCode.doc' file is referred to distinguish the .txt file based on the programming language of the test or application code.

(2) Few of the histogram plots generated are not clear enough to analyze because of the huge data, it becomes difficult to adjust the bins at each step. Thus, for the analysis, the histogram plots are used which are created using the MS Excel tool (In the .csv file, select the column whose histogram graph is to be plotted. Select the insert tab, and from the list of recommended charts, choose Histogram. And it will plot the graph. Edit the title of the graph, if necessary.).

(3) The cyclomatic complexity at class or file level can also be considered in the further study by using SumCyclomatic data present in the metrics files.

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

(4) Case Study [Foundation DB]:

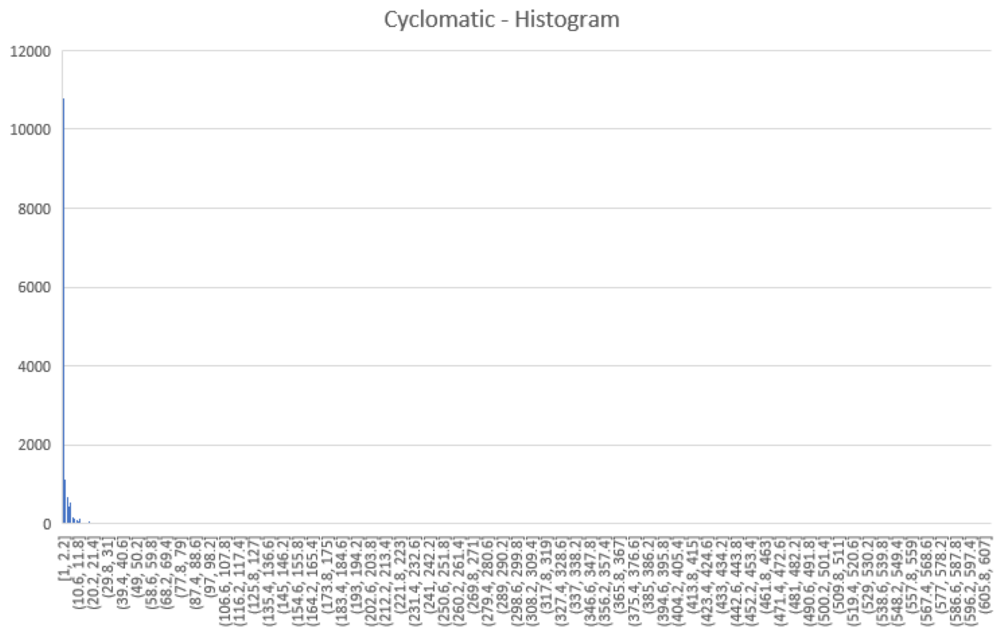


Fig A.4.1 Distribution of CC for different Functions of Application Code (Foundation DB)

	Cyclomatic
count	14728.0
mean	3.012085822922325
std	8.74455800358381
min	1.0
25%	1.0
50%	1.0
75%	3.0
max	610.0

Table A.4.1 Descriptive Statistics of CC for different Functions of Application Code (Foundation DB)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

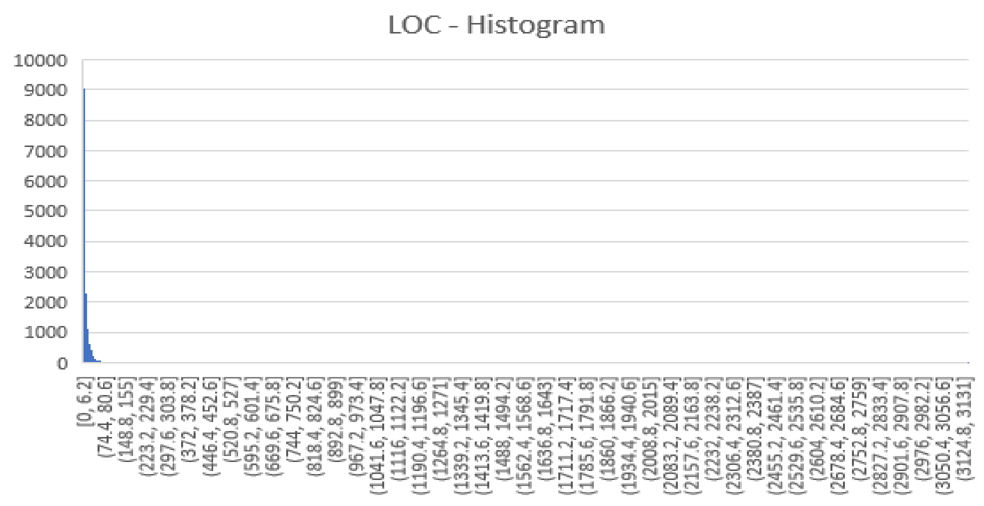
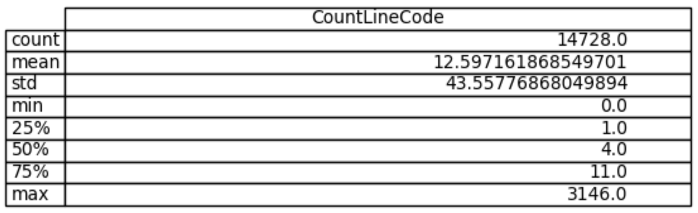


Fig A.4.2 Distribution of LOC for different Functions of Application Code (Foundation DB)



	CountLineCode
count	14728.0
mean	12.597161868549701
std	43.55776868049894
min	0.0
25%	1.0
50%	4.0
75%	11.0
max	3146.0

Table A.4.2 Descriptive Statistics of LOC for different Functions of Application Code (Foundation DB)

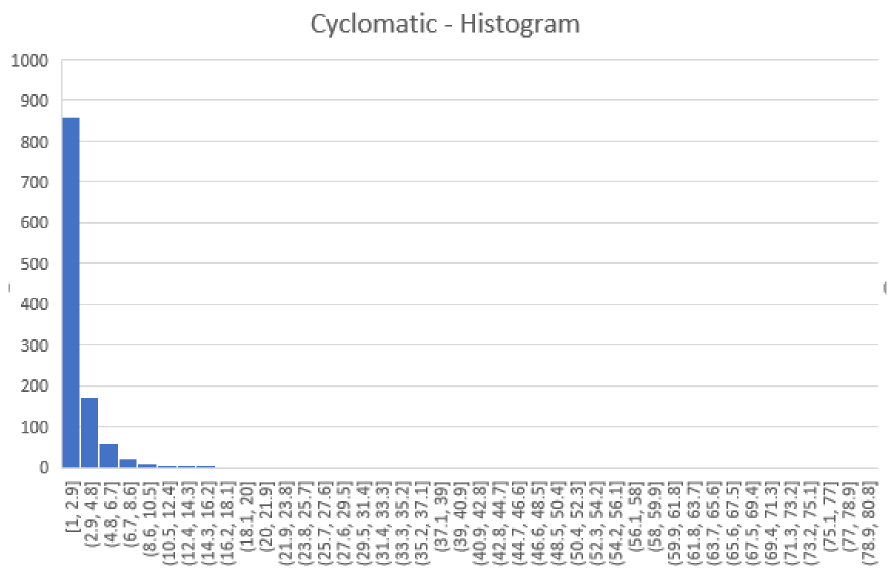
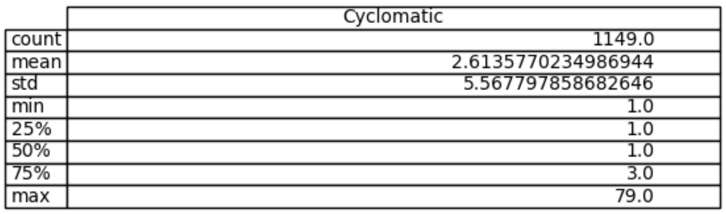


Fig A.4.3 Distribution of CC for different Functions of Test Code (Foundation DB)



	Cyclomatic
count	1149.0
mean	2.6135770234986944
std	5.567797858682646
min	1.0
25%	1.0
50%	1.0
75%	3.0
max	79.0

Table A.4.3 Descriptive Statistics of CC for different Functions of Test Code (Foundation DB)

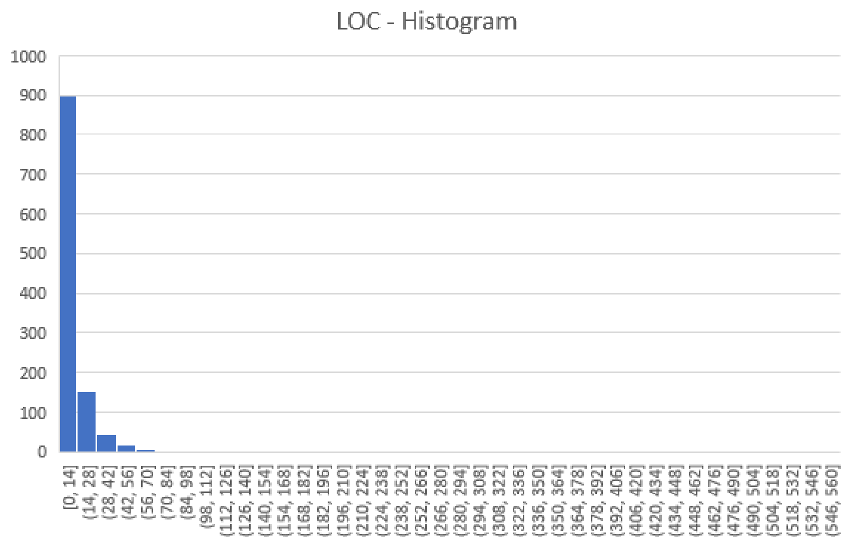
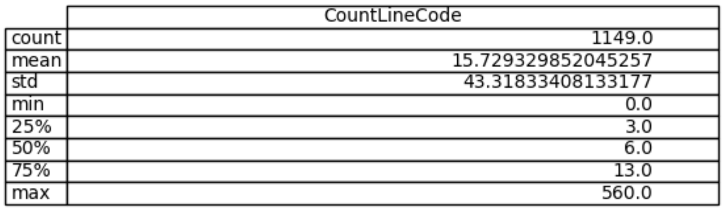


Fig A.4.4 Distribution of LOC for different Functions of Test Code (Foundation DB)



	CountLineCode
count	1149.0
mean	15.729329852045257
std	43.31833408133177
min	0.0
25%	3.0
50%	6.0
75%	13.0
max	560.0

Table A.4.4 Descriptive Statistics of LOC for different Functions of Test Code (Foundation DB)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

(5) Case Study [Curl]:

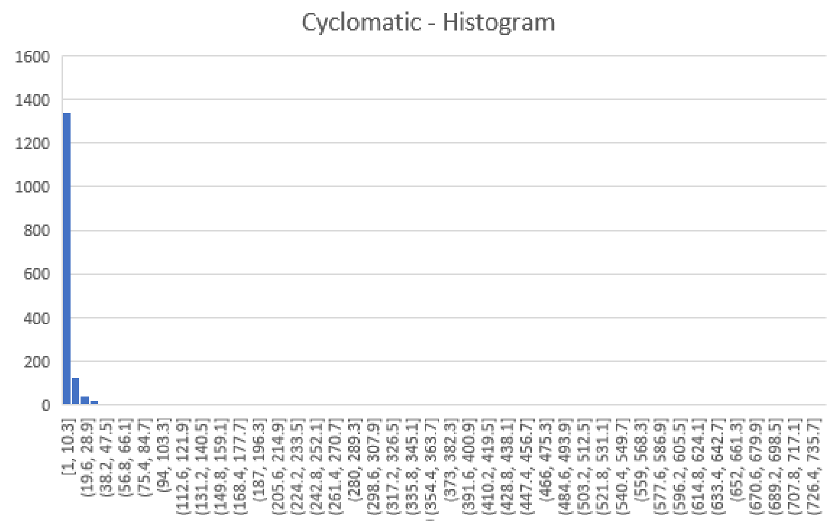


Fig A.5.1 Distribution of CC for different Functions of Application Code (Curl)

	Cyclomatic
count	1573.0
mean	8.231404958677686
std	30.95860086146731
min	1.0
25%	2.0
50%	3.0
75%	7.0
max	740.0

Table A.5.1 Descriptive Statistics of CC for different Functions of Application Code (Curl)

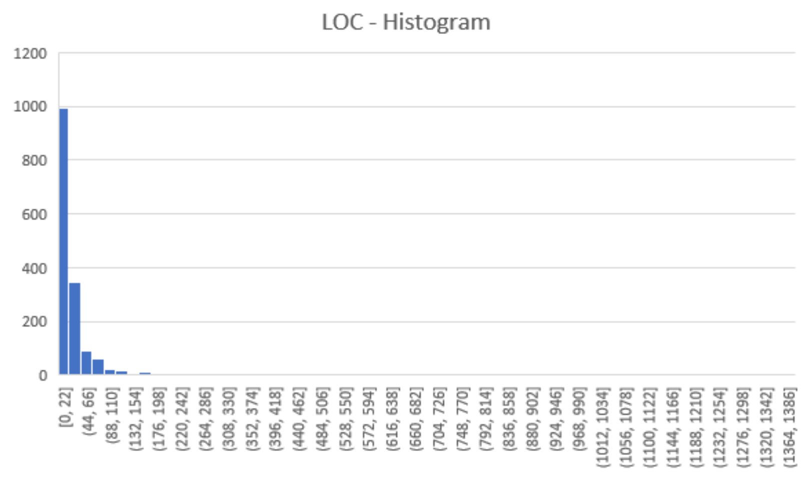
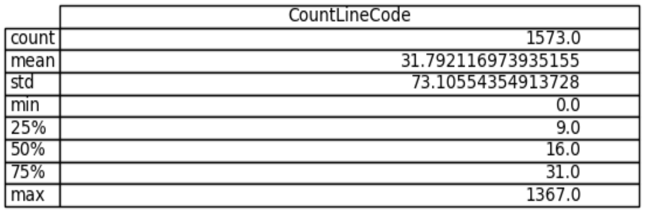


Fig A.5.2 Distribution of LOC for different Functions of Application Code (Curl)



	CountLineCode
count	1573.0
mean	31.792116973935155
std	73.10554354913728
min	0.0
25%	9.0
50%	16.0
75%	31.0
max	1367.0

Table A.5.2 Descriptive Statistics of LOC for different Functions of Application Code (Curl)

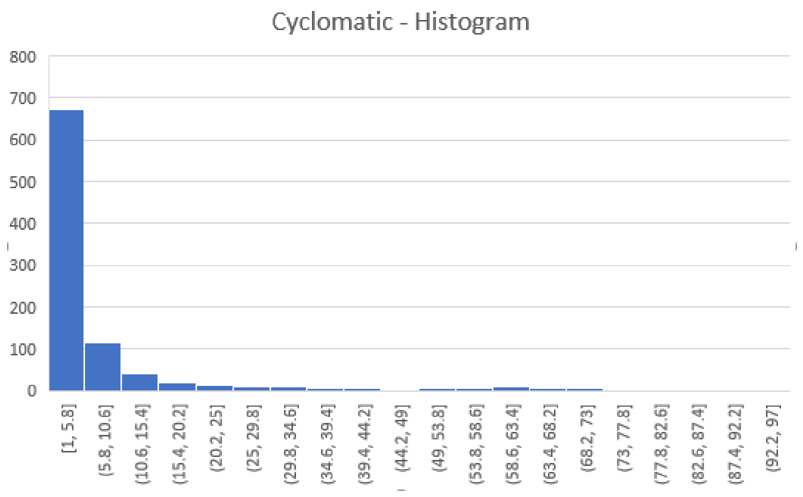
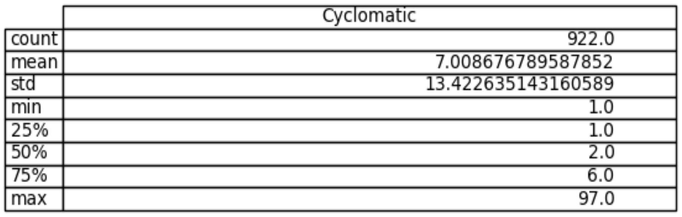


Fig A.5.3 Distribution of CC for different Functions of Test Code (Curl)



	Cyclomatic
count	922.0
mean	7.008676789587852
std	13.422635143160589
min	1.0
25%	1.0
50%	2.0
75%	6.0
max	97.0

Table A.5.3 Descriptive Statistics of CC for different Functions of Test Code (Curl)

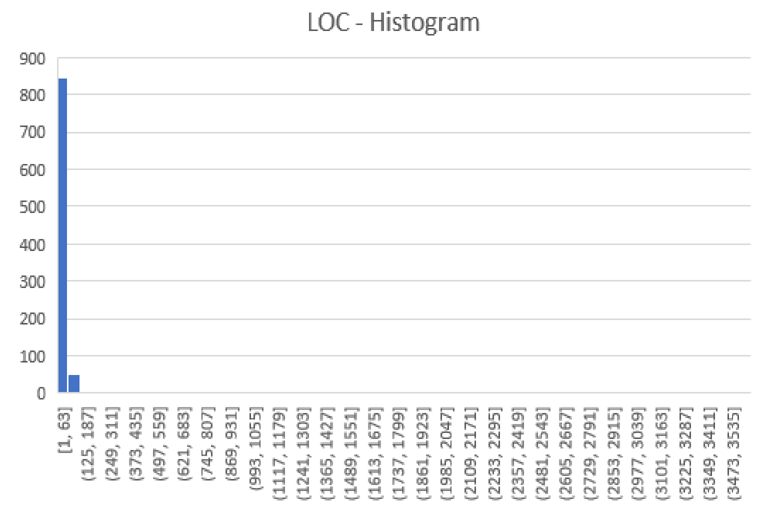


Fig A.5.4 Distribution of LOC for different Functions of Test Code (Curl)

	CountLineCode
count	922.0
mean	33.53036876355748
std	172.6043835922488
min	1.0
25%	3.0
50%	10.0
75%	26.0
max	3570.0

Table A.5.4 Descriptive Statistics of LOC for different Functions of Test Code (Curl)

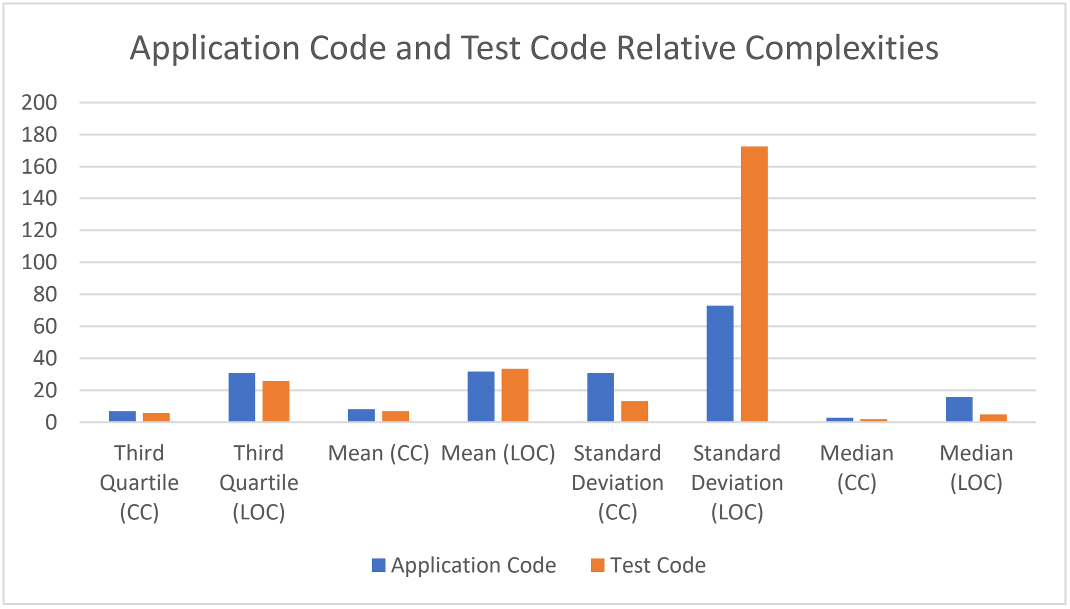


Fig A.5.5 Comparative Statistical Analysis of Application and Test Code Complexities (Curl)

(6) Case Study [Client]:

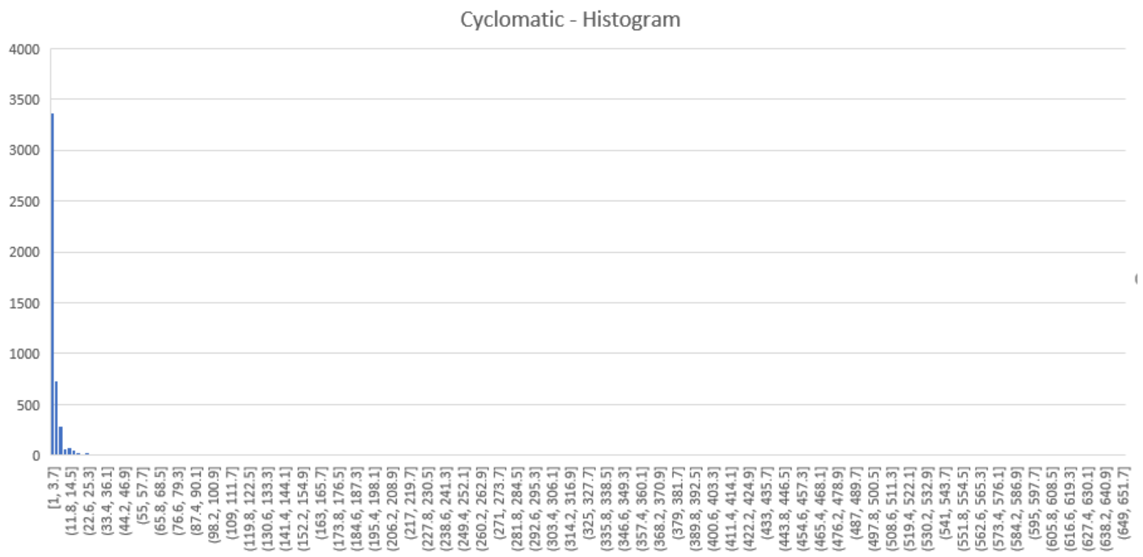
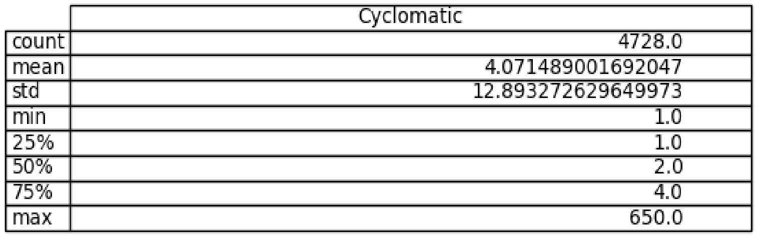


Fig A.6.1 Distribution of CC for different Functions of Application Code (Client)



	Cyclomatic
count	4728.0
mean	4.071489001692047
std	12.893272629649973
min	1.0
25%	1.0
50%	2.0
75%	4.0
max	650.0

Table A.6.1 Descriptive Statistics of CC for different Functions of Application Code (Client)

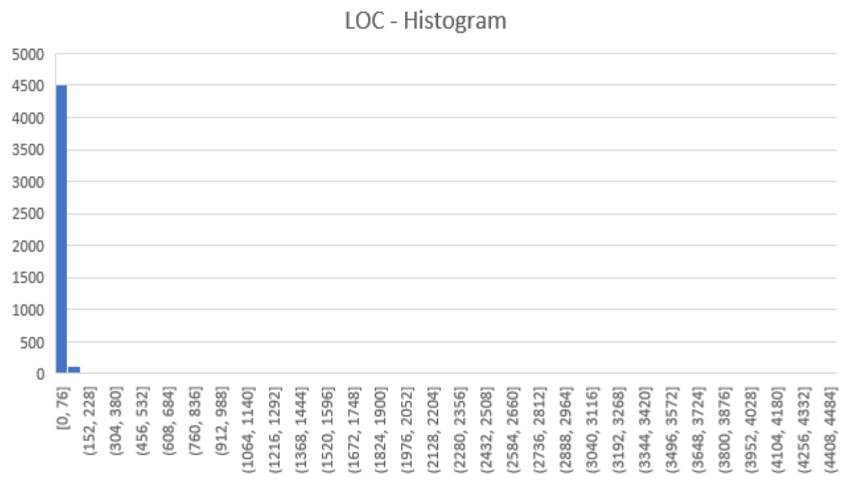
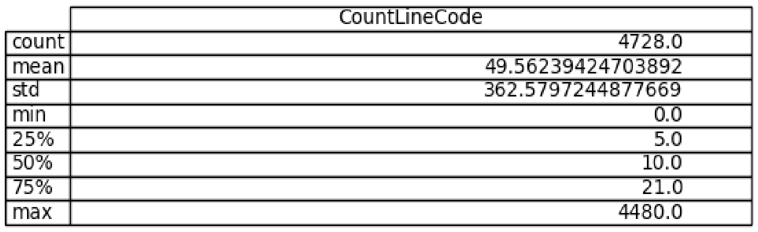


Fig A.6.2 Distribution of LOC for different Functions of Application Code (Client)



	CountLineCode
count	4728.0
mean	49.56239424703892
std	362.5797244877669
min	0.0
25%	5.0
50%	10.0
75%	21.0
max	4480.0

Table A.6.2 Descriptive Statistics of LOC for different Functions of Application Code (Client)

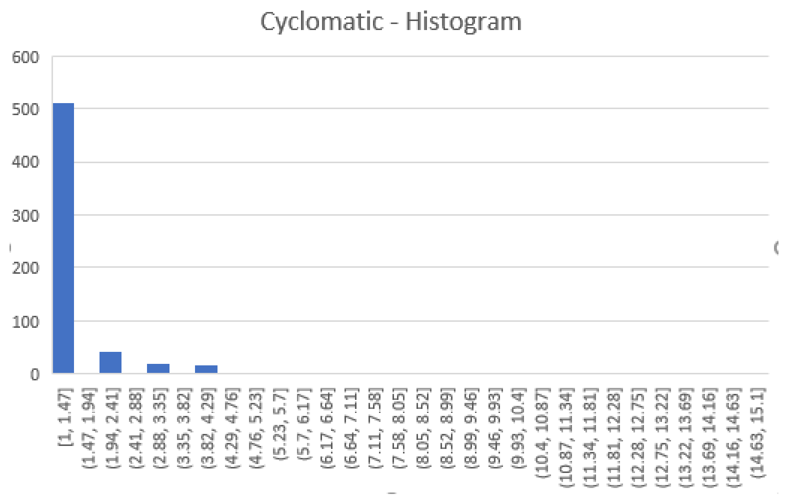
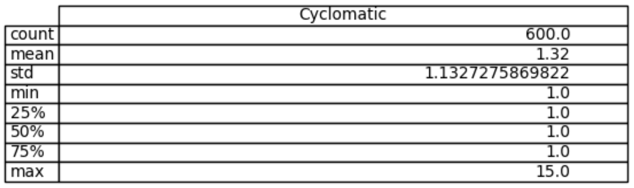


Fig A.6.3 Distribution of CC for different Functions of Test Code (Client)



	Cyclomatic
count	600.0
mean	1.32
std	1.1327275869822
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	15.0

Table A.6.3 Descriptive Statistics of CC for different Functions of Test Code (Client)

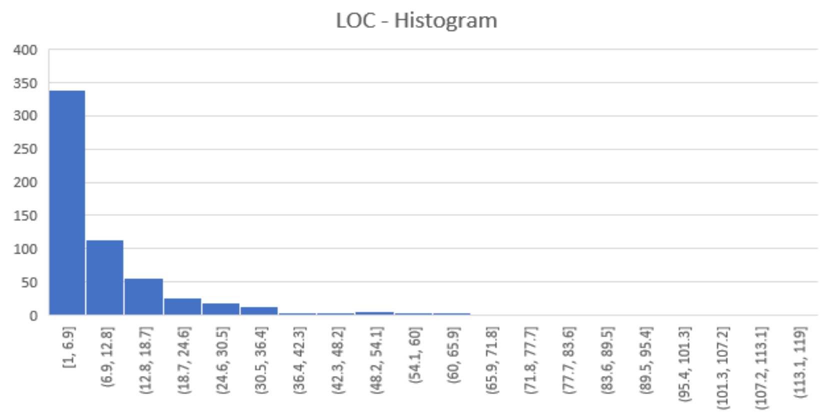
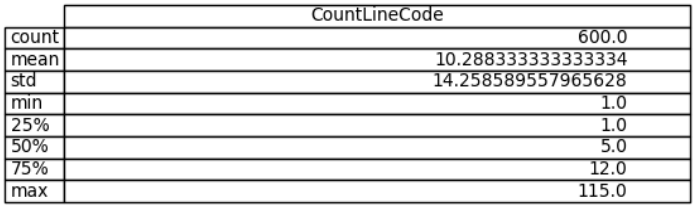


Fig A.6.4 Distribution of LOC for different Functions of Test Code (Client)



	CountLineCode
count	600.0
mean	10.288333333333334
std	14.258589557965628
min	1.0
25%	1.0
50%	5.0
75%	12.0
max	115.0

Table A.6.4 Descriptive Statistics of LOC for different Functions of Test Code (Client)

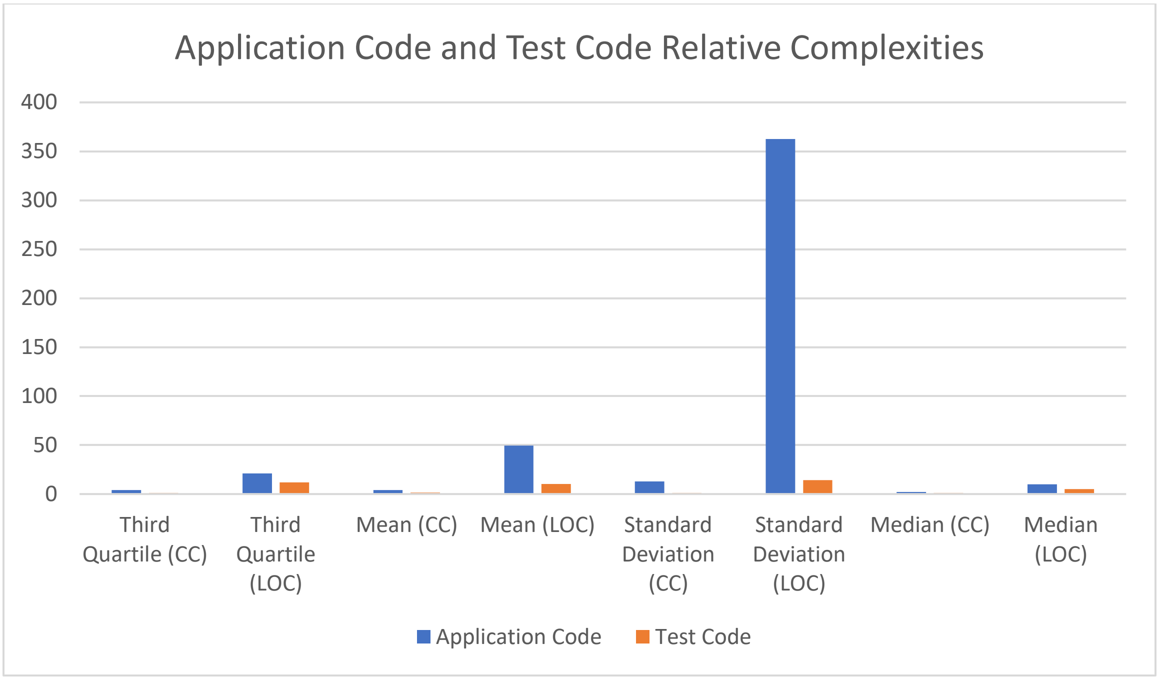


Fig A.6.5 Comparative Statistical Analysis of Application and Test Code Complexities (Client)

(7) Case Study [MuseScore]:

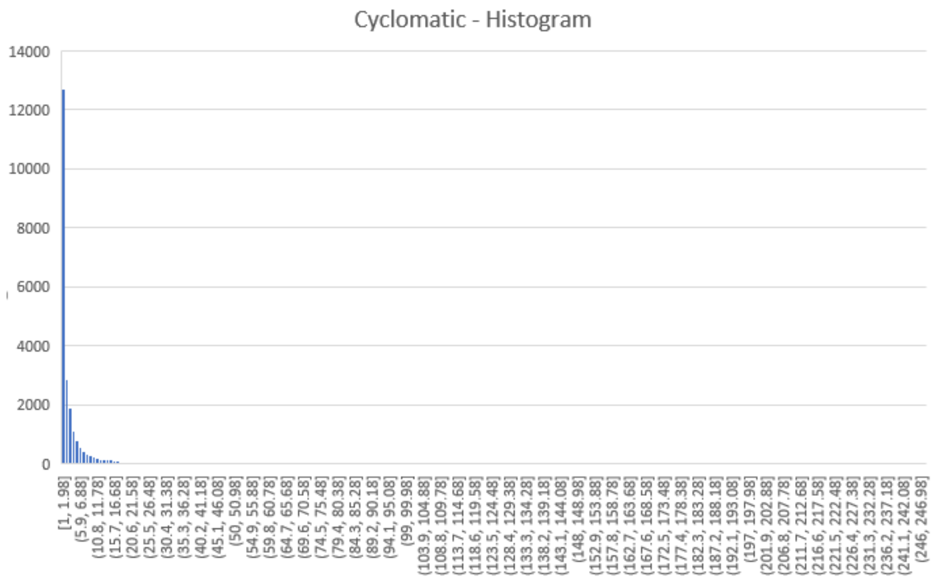
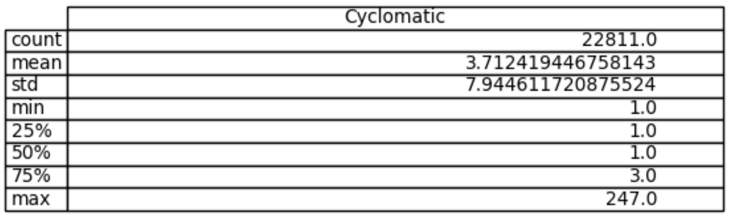


Fig A.7.1 Distribution of CC for different Functions of Application Code (Muse Score)



	Cyclomatic
count	22811.0
mean	3.712419446758143
std	7.944611720875524
min	1.0
25%	1.0
50%	1.0
75%	3.0
max	247.0

Table A.7.1 Descriptive Statistics of CC for different Functions of Application Code (Muse Score)

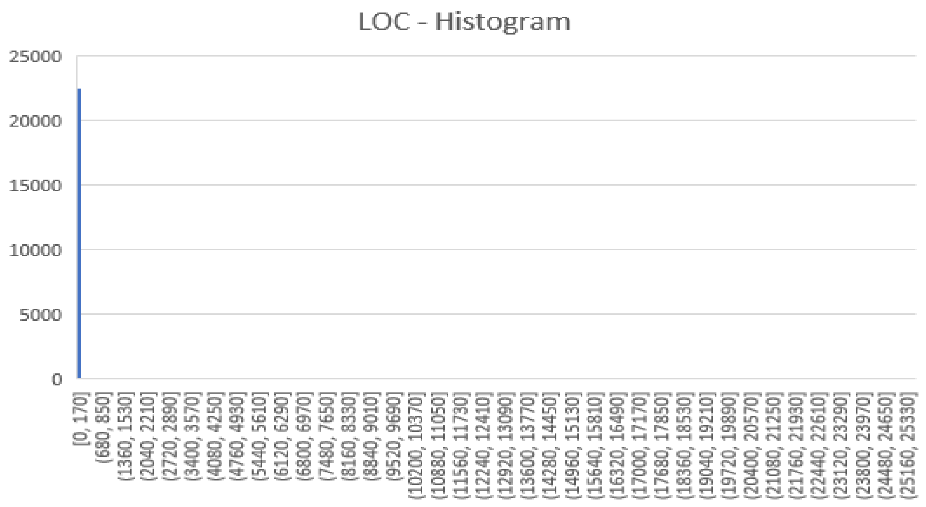
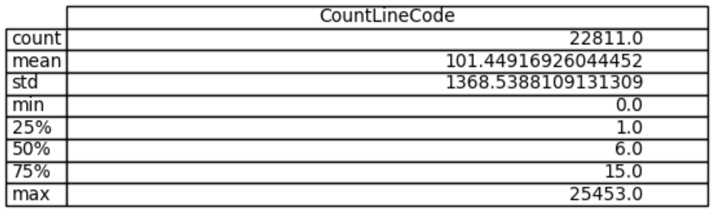


Fig A.7.2 Distribution of LOC for different Functions of Application Code (Muse Score)



	CountLineCode
count	22811.0
mean	101.44916926044452
std	1368.5388109131309
min	0.0
25%	1.0
50%	6.0
75%	15.0
max	25453.0

Table A.7.2 Descriptive Statistics of LOC for different Functions of Application Code (Muse Score)

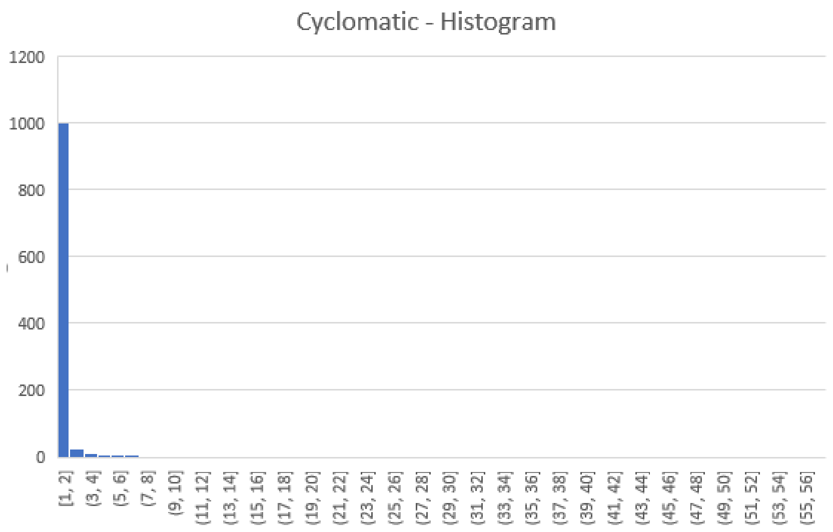
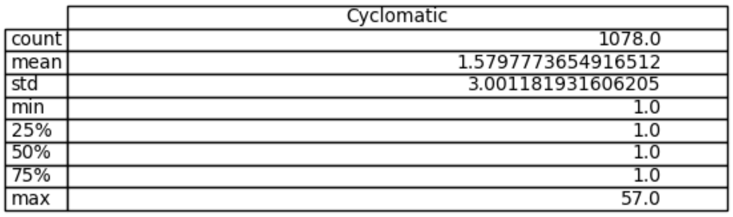


Fig A.7.3 Distribution of CC for different Functions of Test Code (Muse Score)



	Cyclomatic
count	1078.0
mean	1.5797773654916512
std	3.001181931606205
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	57.0

Table A.7.3 Descriptive Statistics of CC for different Functions of Test Code (Muse Score)

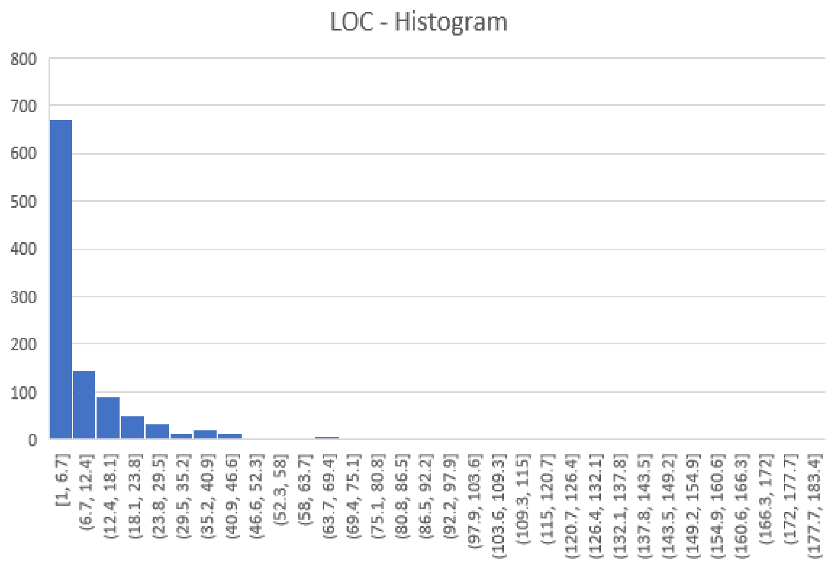
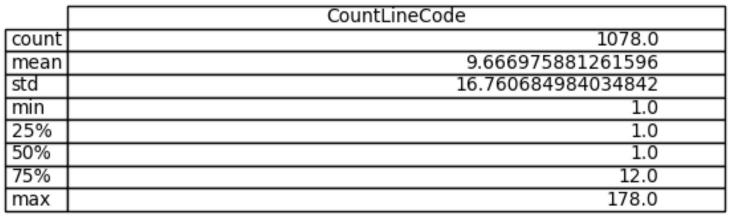


Fig A.7.4 Distribution of LOC for different Functions of Test Code (Muse Score)



	CountLineCode
count	1078.0
mean	9.666975881261596
std	16.760684984034842
min	1.0
25%	1.0
50%	1.0
75%	12.0
max	178.0

Table A.7.4 Descriptive Statistics of LOC for different Functions of Test Code (Muse Score)

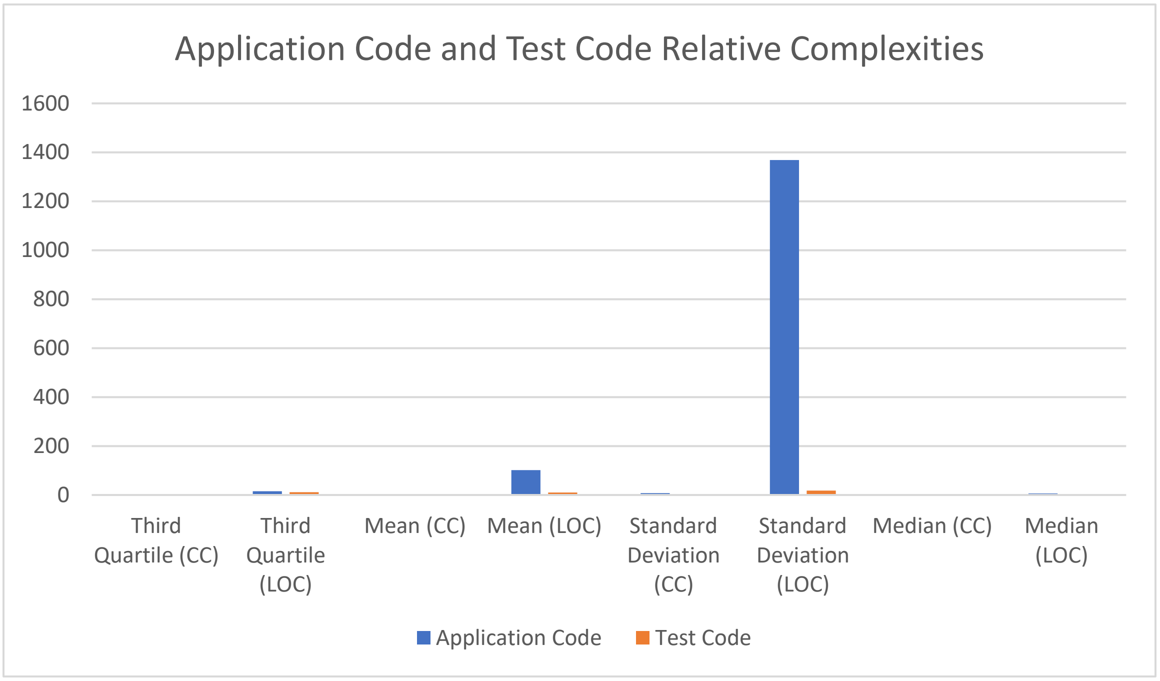


Fig A.7.5 Comparative Statistical Analysis of Application and Test Code Complexities (Muse Score)

(8) Case Study [Scintilla-Notepad]:

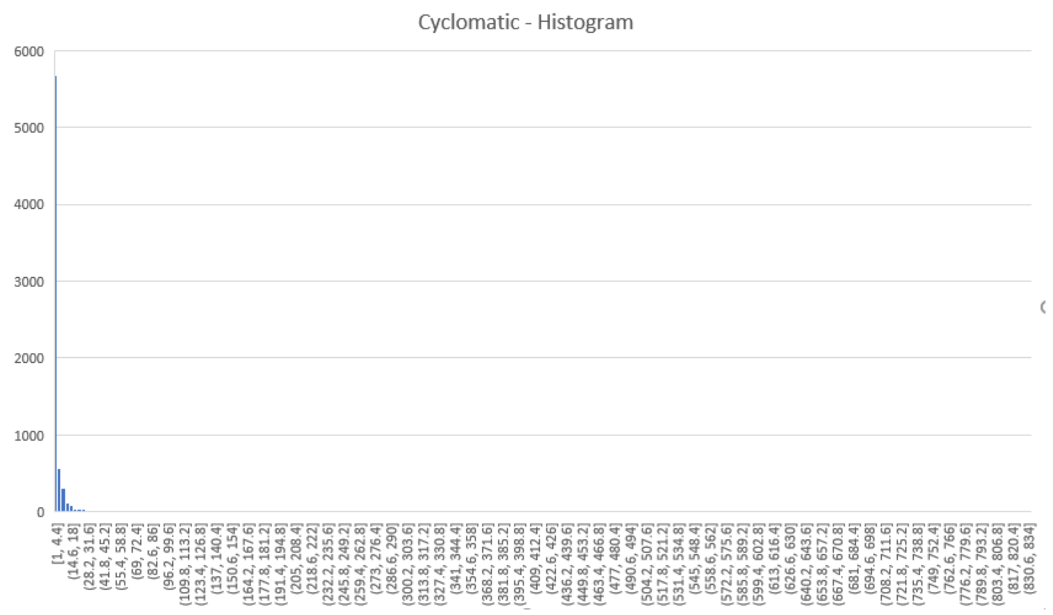
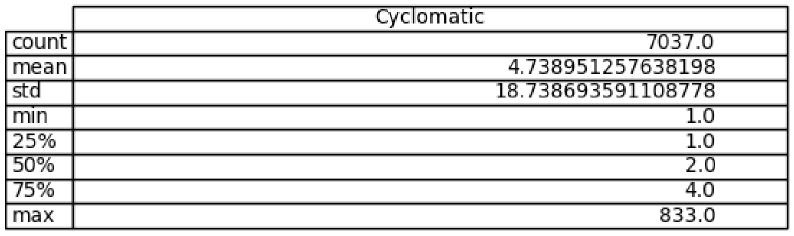


Fig A.8.1 Distribution of CC for different Functions of Application Code (Scintilla-Notepad)



	Cyclomatic
count	7037.0
mean	4.738951257638198
std	18.738693591108778
min	1.0
25%	1.0
50%	2.0
75%	4.0
max	833.0

Table A.8.1 Descriptive Statistics of CC for different Functions of Application Code (Scintilla-Notepad)

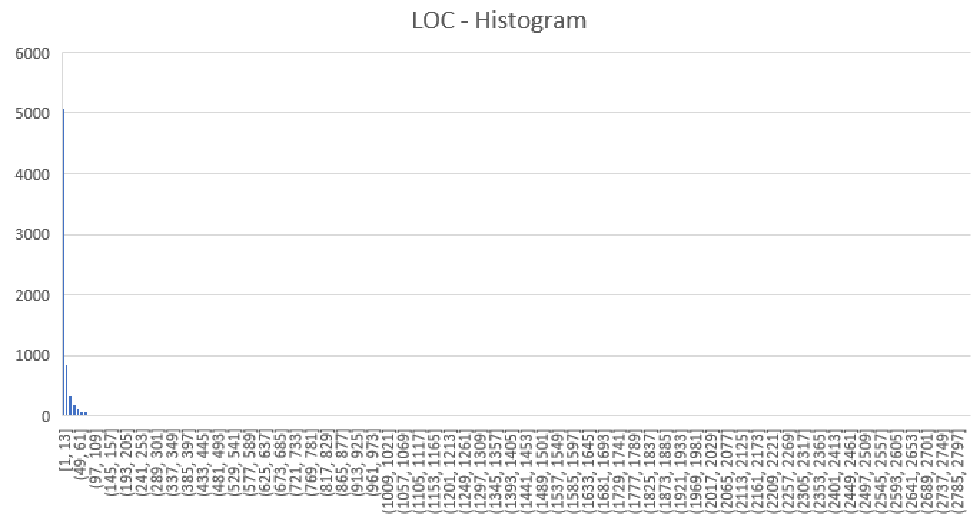
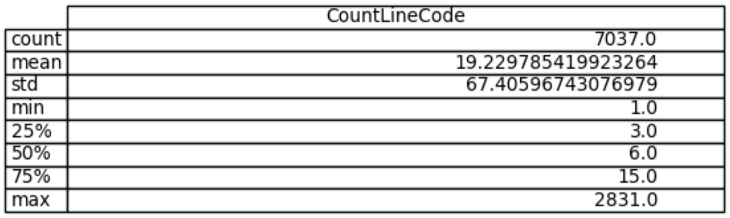


Fig A.8.2 Distribution of LOC for different Functions of Application Code (Scintilla-Notepad)



	CountLineCode
count	7037.0
mean	19.229785419923264
std	67.40596743076979
min	1.0
25%	3.0
50%	6.0
75%	15.0
max	2831.0

Table A.8.2 Descriptive Statistics of LOC for different Functions of Application Code (Scintilla-Notepad)

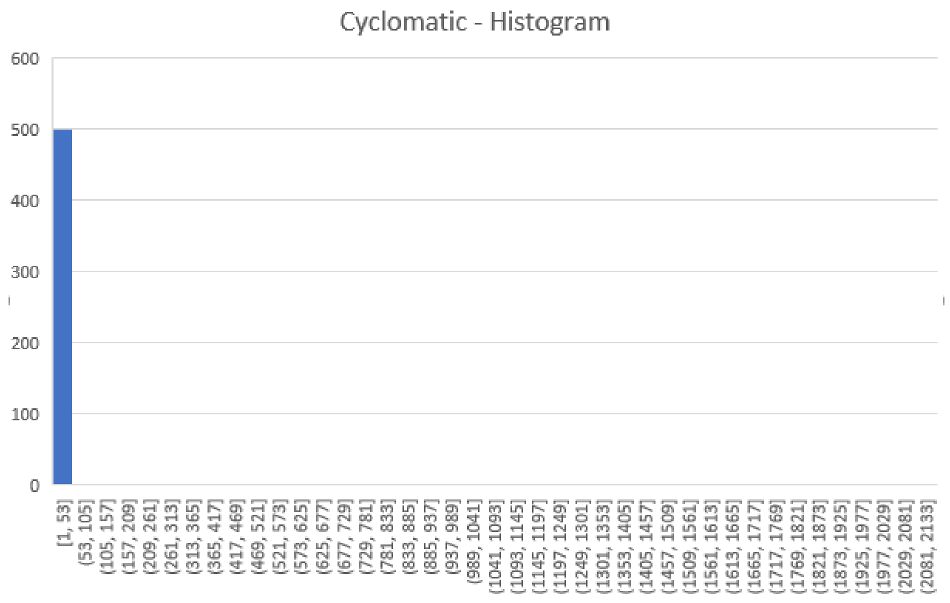
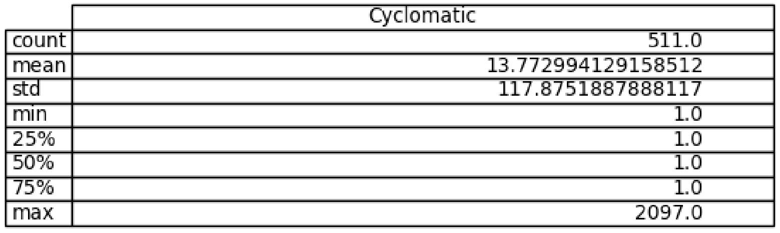


Fig A.8.3 Distribution of CC for different Functions of Test Code (Scintilla-Notepad)



	Cyclomatic
count	511.0
mean	13.772994129158512
std	117.8751887888117
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	2097.0

Table A.8.3 Descriptive Statistics of CC for different Functions of Test Code (Scintilla-Notepad)

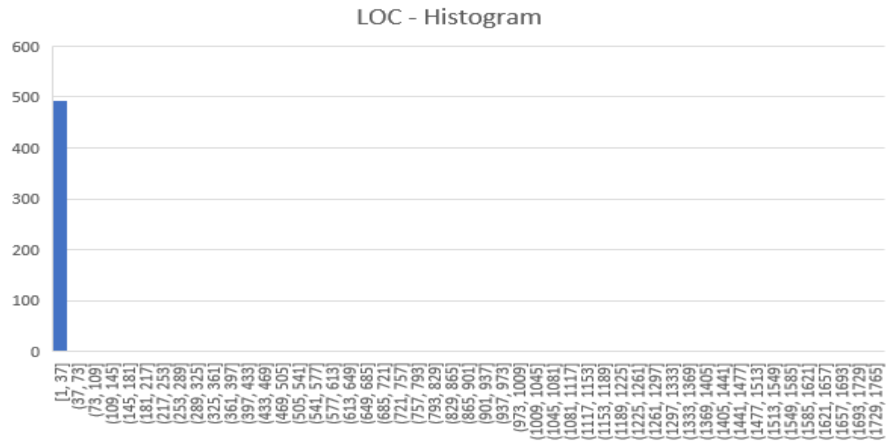
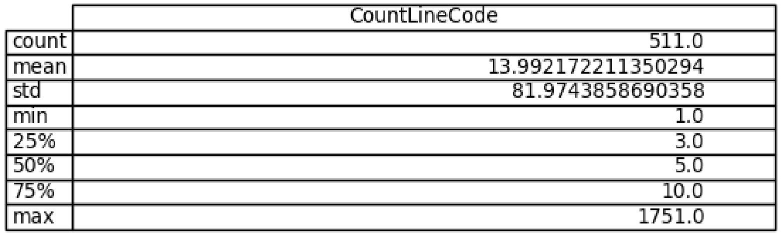


Fig A.8.4 Distribution of LOC for different Functions of Test Code (Scintilla-Notepad)



	CountLineCode
count	511.0
mean	13.992172211350294
std	81.9743858690358
min	1.0
25%	3.0
50%	5.0
75%	10.0
max	1751.0

Table A.8.4 Descriptive Statistics of LOC for different Functions of Test Code (Scintilla-Notepad)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

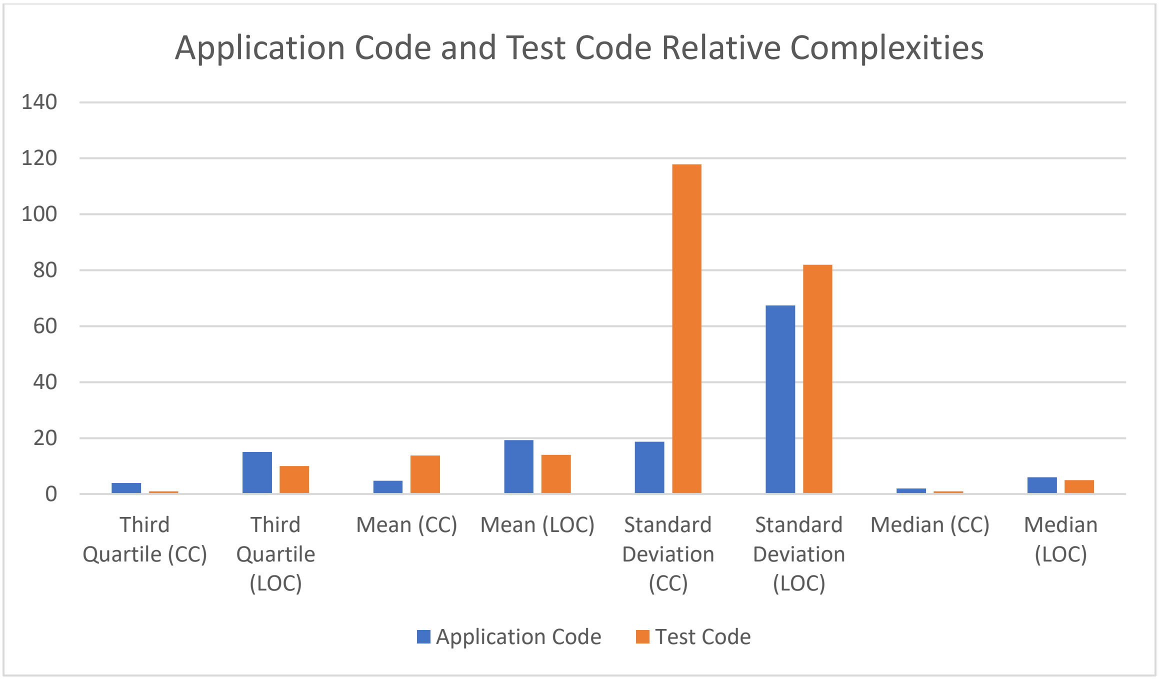


Fig A.8.5 Comparative Statistical Analysis of Application and Test Code Complexities (Scintilla)

(9) Case Study [JRuby]:

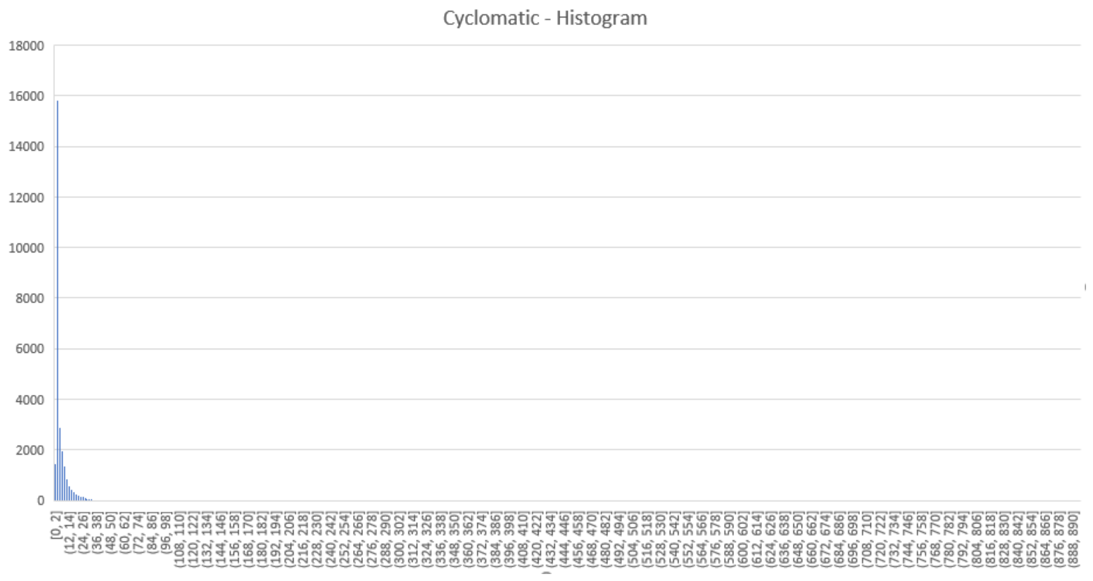


Fig A.9.1 Distribution of CC for different Functions of Application Code (JRuby)

	Cyclomatic
count	27394.0
mean	2.0203329196174344
std	4.139438320139762
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	259.0

Table A.9.1 Descriptive Statistics of CC for different Functions of Application Code (JRuby)

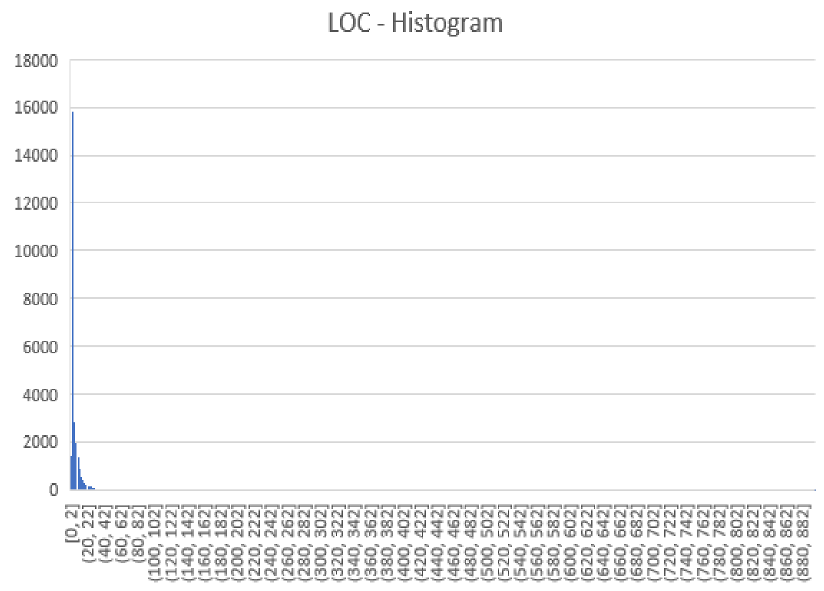


Fig A.9.2 Distribution of LOC for different Functions of Application Code (JRuby)

	CountLineCode
count	27394.0
mean	7.09549536394831
std	17.00352477747185
min	0.0
25%	3.0
50%	3.0
75%	7.0
max	895.0

Table A.9.2 Descriptive Statistics of LOC for different Functions of Application Code (JRuby)

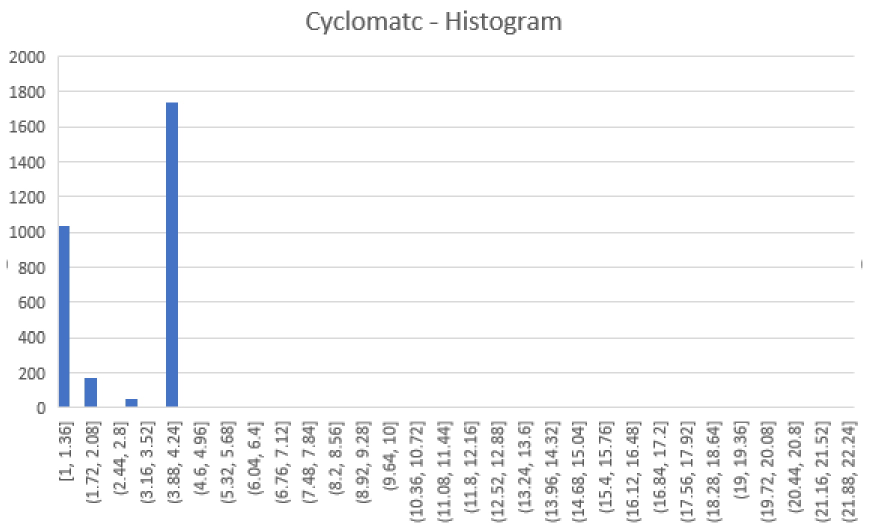
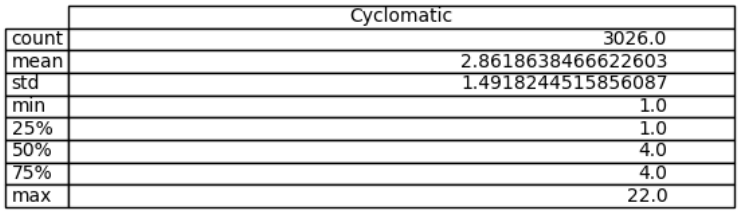


Fig A.9.3 Distribution of CC for different Functions of Test Code (JRuby)



	Cyclomatic
count	3026.0
mean	2.8618638466622603
std	1.4918244515856087
min	1.0
25%	1.0
50%	4.0
75%	4.0
max	22.0

Table A.9.3 Descriptive Statistics of CC for different Functions of Test Code (JRuby)

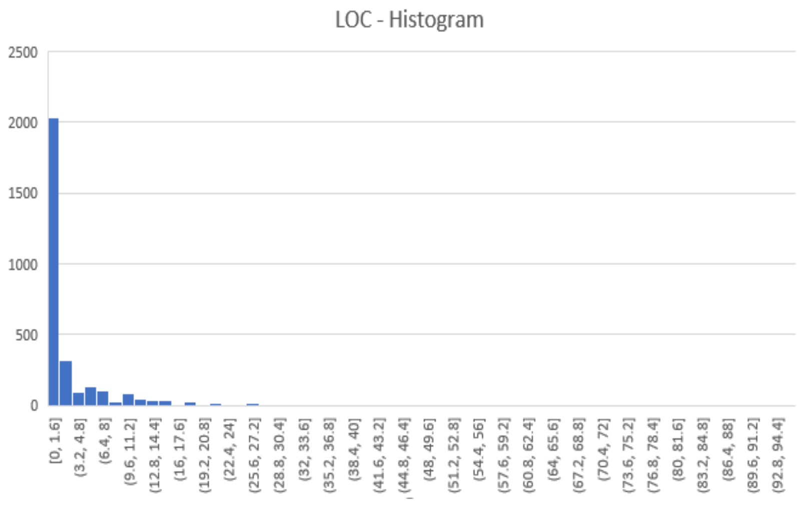
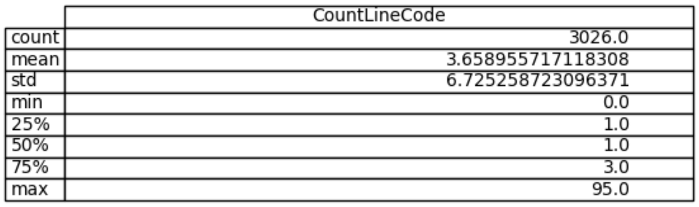


Fig A.9.4 Distribution of LOC for different Functions of Test Code (JRuby)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca



	CountLineCode
count	3026.0
mean	3.658955717118308
std	6.725258723096371
min	0.0
25%	1.0
50%	1.0
75%	3.0
max	95.0

Table A.9.4 Descriptive Statistics of LOC for different Functions of Test Code (JRuby)

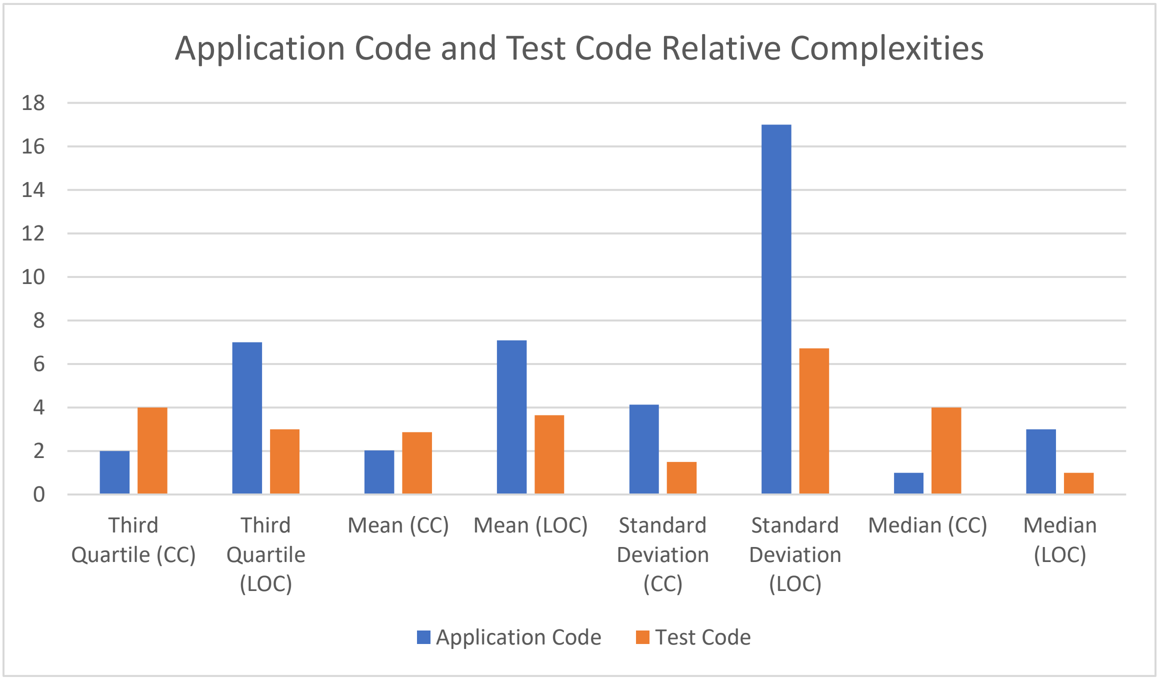


Fig A.9.5 Comparative Statistical Analysis of Application and Test Code Complexities (JRuby)

(10) Case Study [Abiword]:

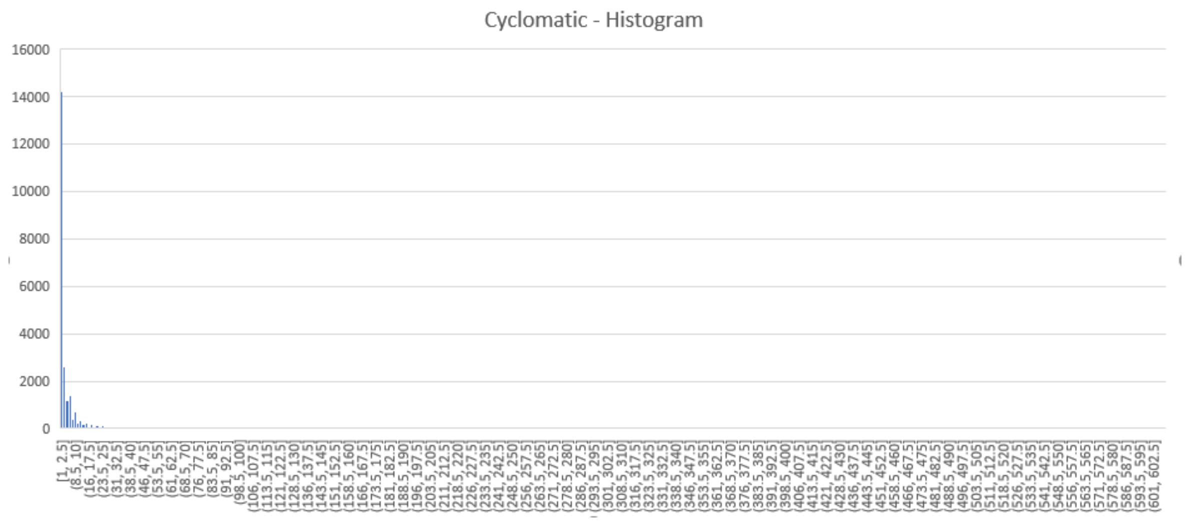
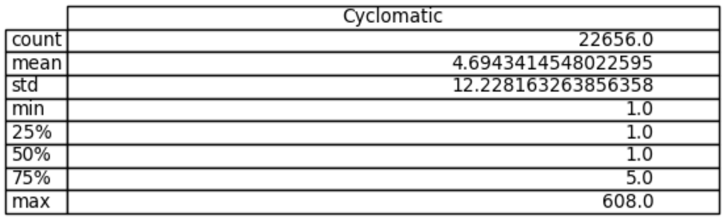


Fig A.10.1 Distribution of CC for different Functions of Application Code (Abiword)



	Cyclomatic
count	22656.0
mean	4.6943414548022595
std	12.228163263856358
min	1.0
25%	1.0
50%	1.0
75%	5.0
max	608.0

Table A.10.1 Descriptive Statistics of CC for different Functions of Application Code (Abiword)

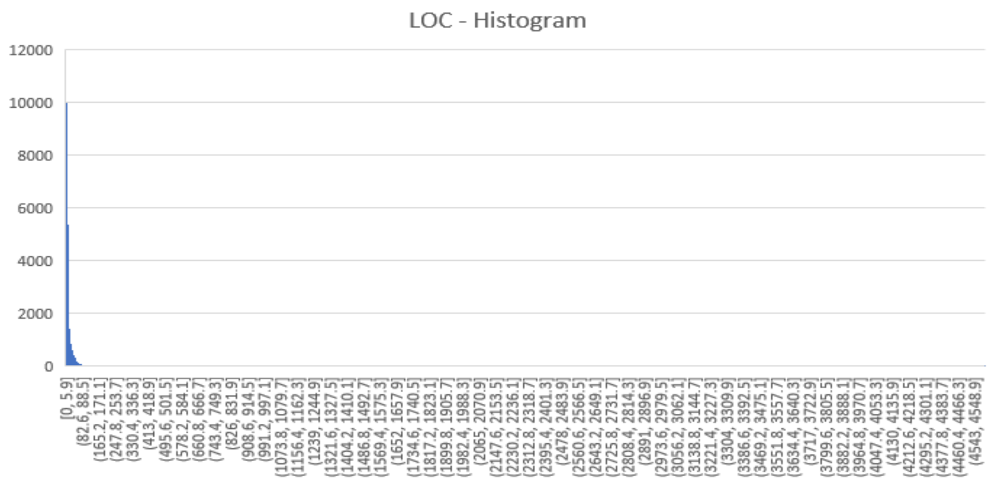
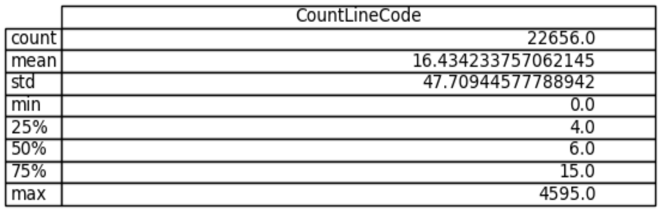


Fig A.10.2 Distribution of LOC for different Functions of Application Code (Abiword)



	CountLineCode
count	22656.0
mean	16.434233757062145
std	47.70944577788942
min	0.0
25%	4.0
50%	6.0
75%	15.0
max	4595.0

Table A.10.2 Descriptive Statistics of LOC for different Functions of Application Code (Abiword)

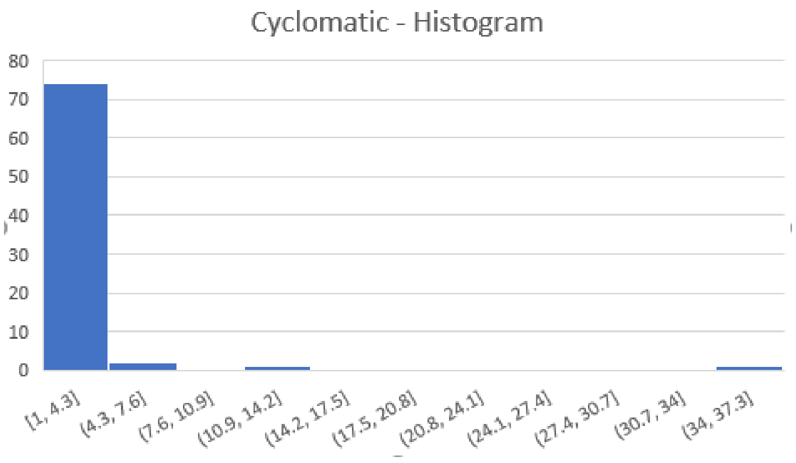
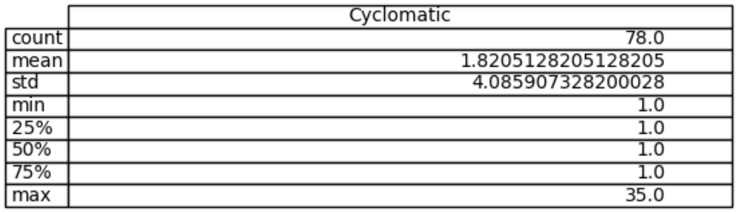


Fig A.10.3 Distribution of CC for different Functions of Test Code (Abiword)



	Cyclomatic
count	78.0
mean	1.8205128205128205
std	4.085907328200028
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	35.0

Table A.10.3 Descriptive Statistics of CC for different Functions of Test Code (Abiword)

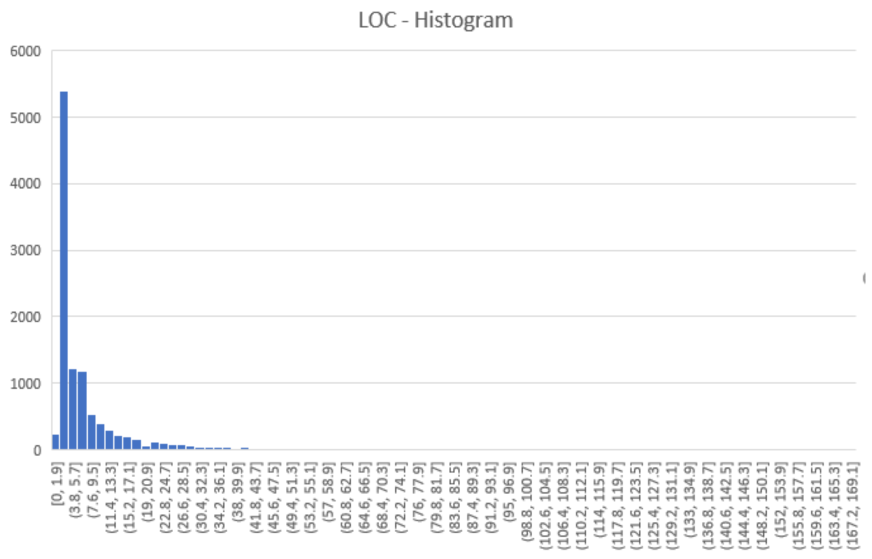
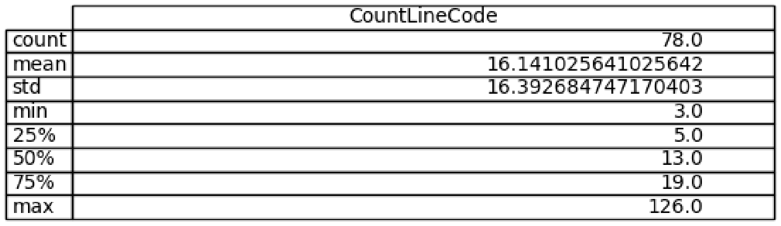


Fig A.10.4 Distribution of LOC for different Functions of Test Code (Abiword)



	CountLineCode
count	78.0
mean	16.141025641025642
std	16.392684747170403
min	3.0
25%	5.0
50%	13.0
75%	19.0
max	126.0

Table A.10.4 Descriptive Statistics of LOC for different Functions of Test Code (Abiword)

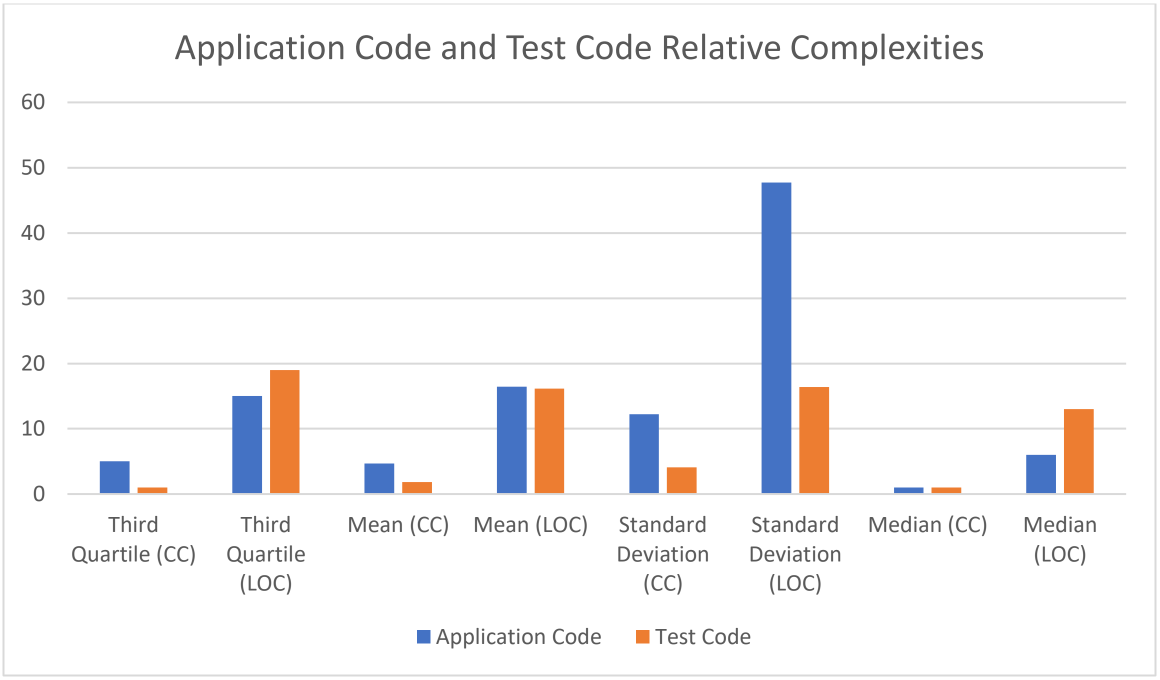


Fig A.10.5 Comparative Statistical Analysis of Application and Test Code Complexities (Abiword)

(11) Case Study [Jkarta JMeter]:

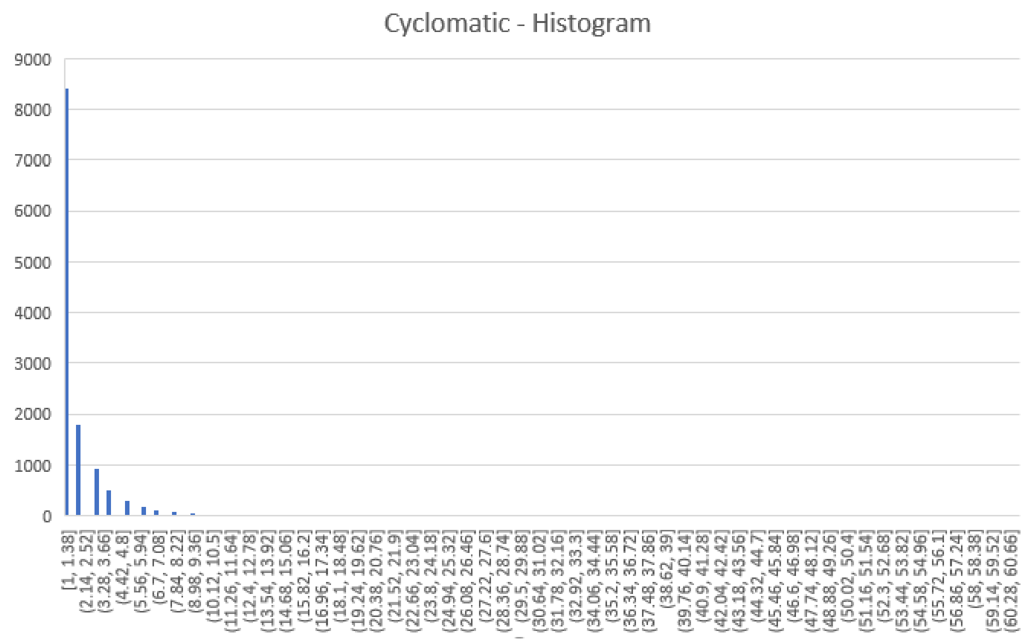


Fig A.11.1 Distribution of CC for different Functions of Application Code (Jkarta JMeter)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

	Cyclomatic
count	12746.0
mean	2.0218107641613057
std	2.550085754718699
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	61.0

Table A.11.1 Descriptive Statistics of CC for different Functions of Application Code (Jkarta JMeter)

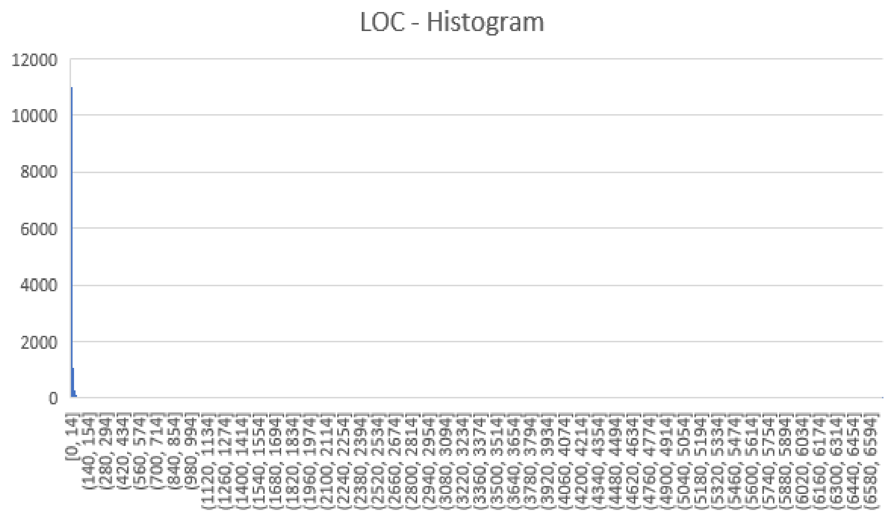


Fig A.11.2 Distribution of LOC for different Functions of Application Code (Jkarta JMeter)

	CountLineCode
count	12746.0
mean	10.308567393692138
std	93.70254908662179
min	0.0
25%	3.0
50%	3.0
75%	8.0
max	6686.0

Table A.11.2 Descriptive Statistics of LOC for different Functions of Application Code (Jkarta JMeter)

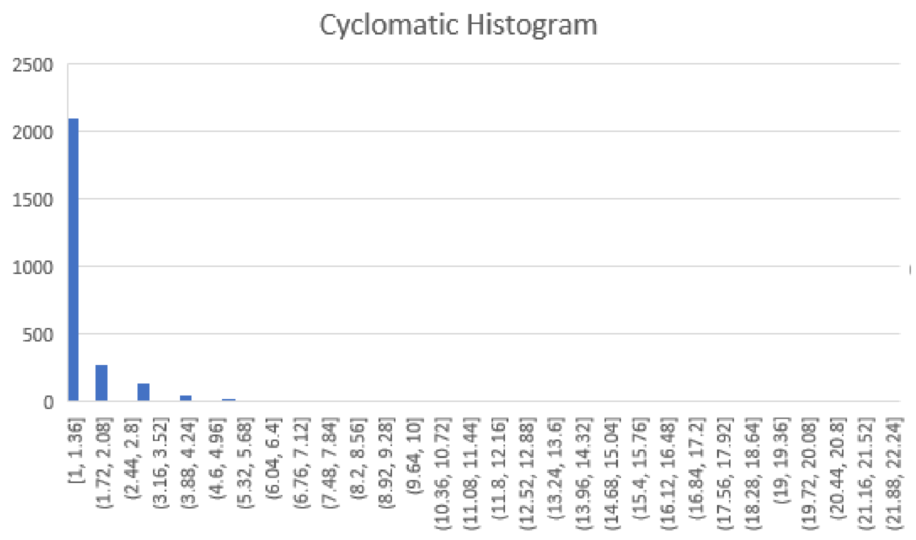
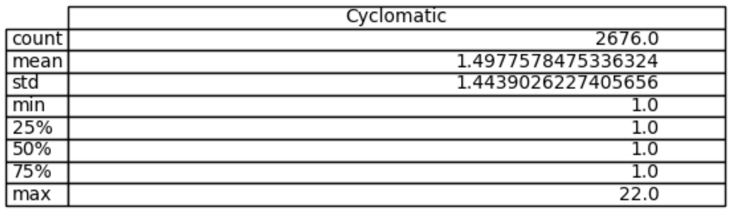


Fig A.11.3 Distribution of CC for different Functions of Test Code (Jkarta JMeter)



	Cyclomatic
count	2676.0
mean	1.4977578475336324
std	1.4439026227405656
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	22.0

Table A.11.3 Descriptive Statistics of CC for different Functions of Test Code (Jkarta JMeter)

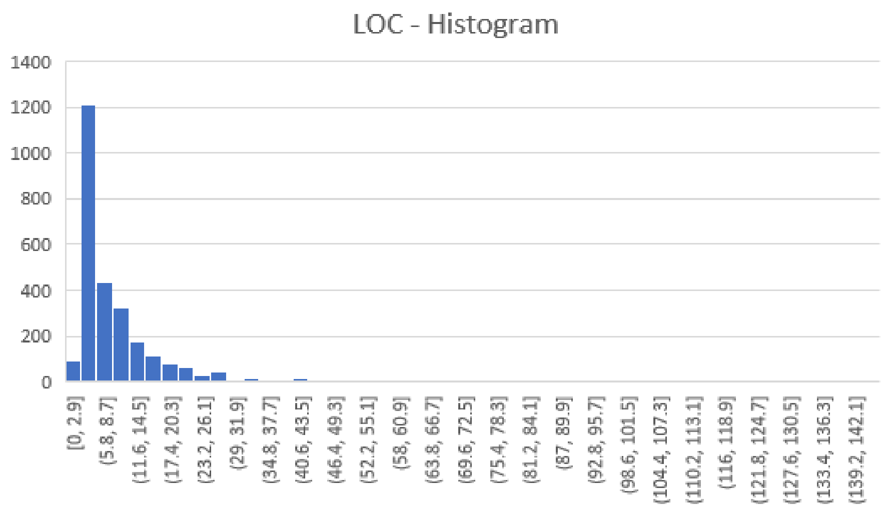
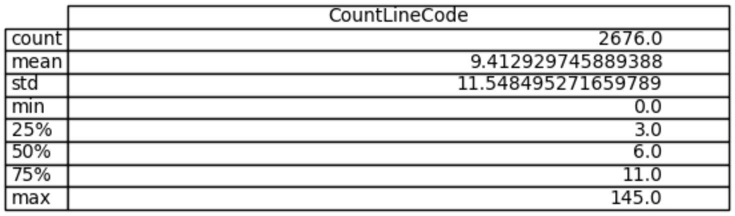


Fig A.11.4 Distribution of LOC for different Functions of Test Code (Jkarta JMeter)



	CountLineCode
count	2676.0
mean	9.412929745889388
std	11.548495271659789
min	0.0
25%	3.0
50%	6.0
75%	11.0
max	145.0

Table A.11.4 Descriptive Statistics of LOC for different Functions of Test Code (Jkarta JMeter)

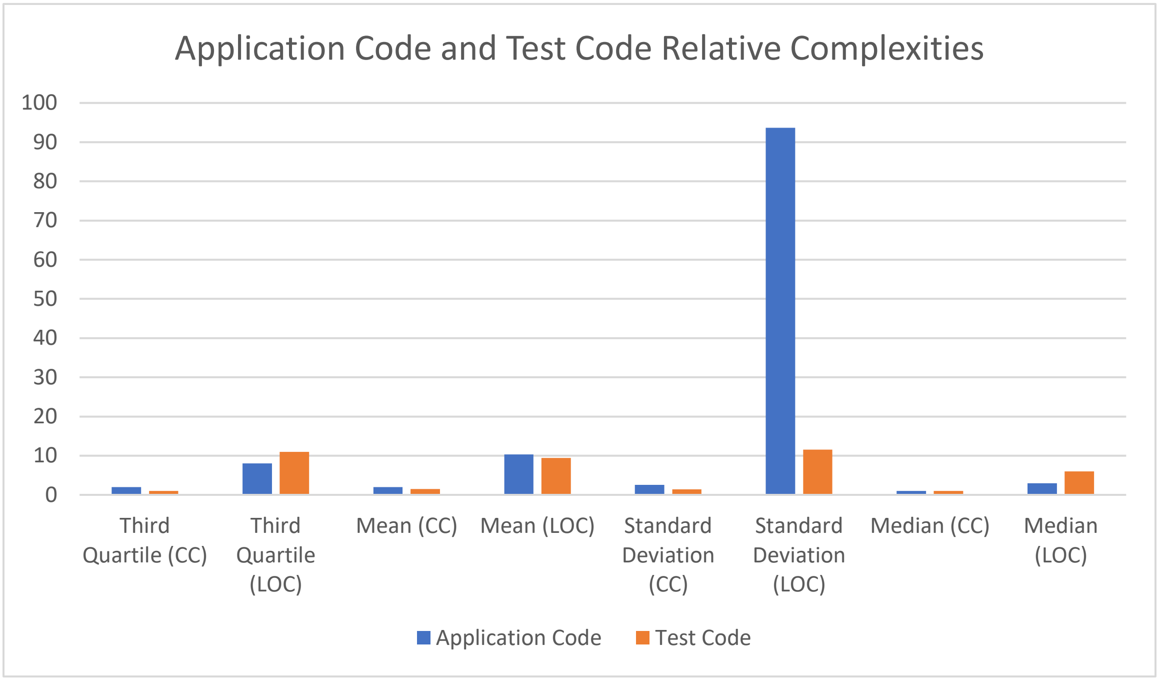


Fig A.11.5 Comparative Statistical Analysis of Application and Test Code Complexities (Jkarta JMeter)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

(12) Case Study [Keepass]:

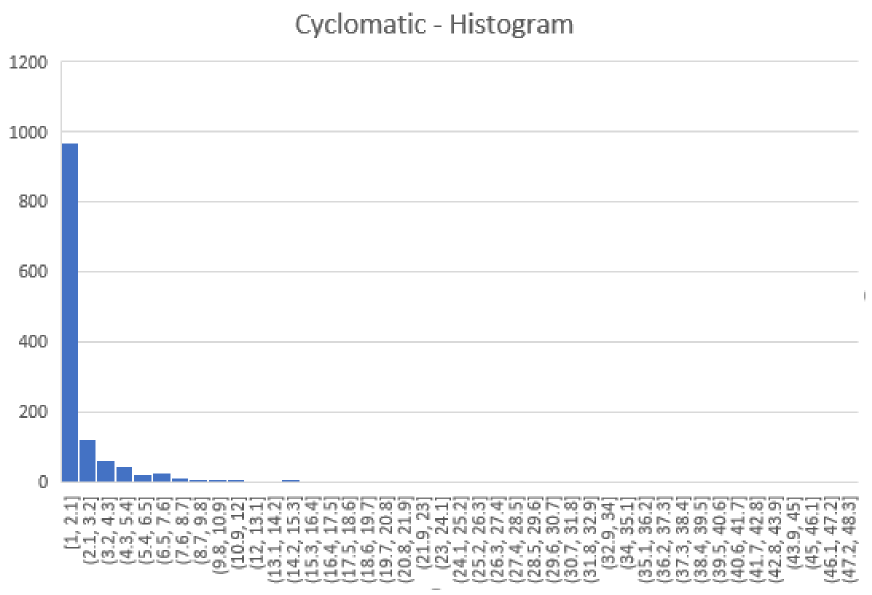
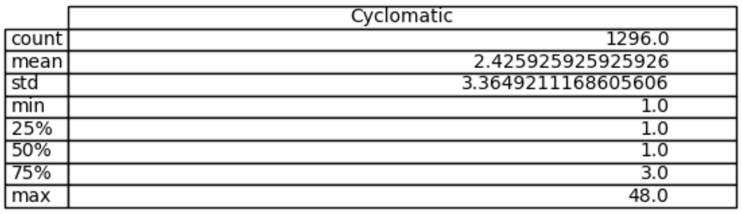


Fig A.12.1 Distribution of CC for different Functions of Application Code (Keepass)



	Cyclomatic
count	1296.0
mean	2.425925925925926
std	3.3649211168605606
min	1.0
25%	1.0
50%	1.0
75%	3.0
max	48.0

Table A.12.1 Descriptive Statistics of CC for different Functions of Application Code (Keepass)

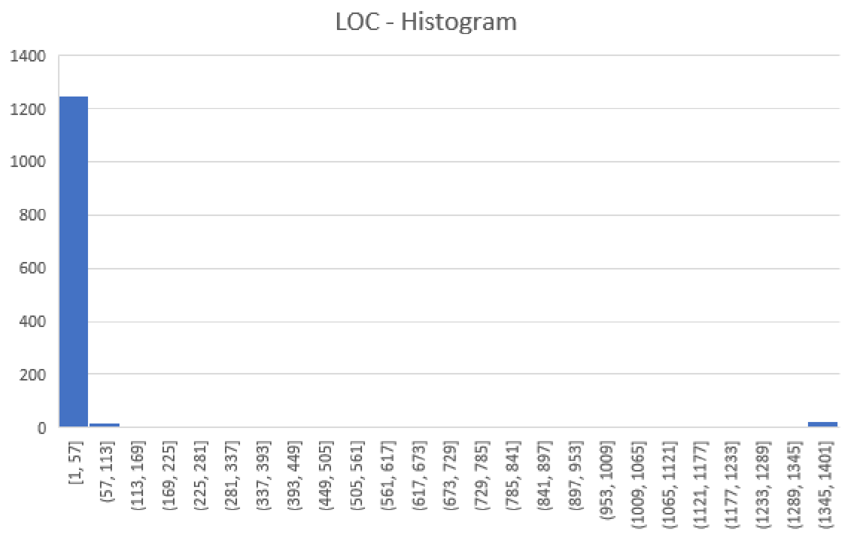


Fig A.12.2 Distribution of LOC for different Functions of Application Code (Keepass)

	CountLineCode
count	1296.0
mean	34.79706790123457
std	174.52232353554538
min	1.0
25%	4.0
50%	7.0
75%	14.0
max	1385.0

Table A.12.2 Descriptive Statistics of LOC for different Functions of Application Code (Keepass)

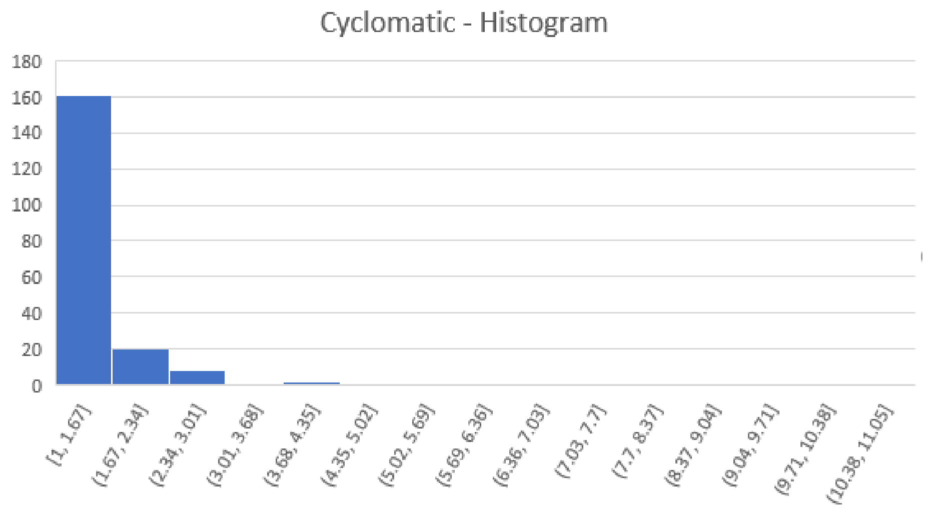
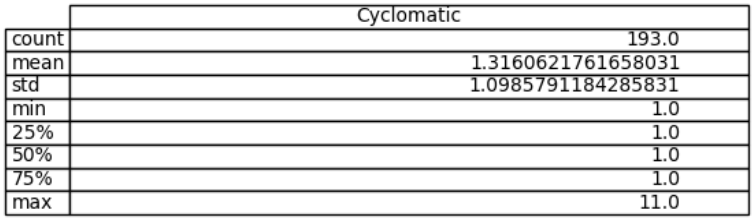


Fig A.12.3 Distribution of CC for different Functions of Test Code (Keepass)



	Cyclomatic
count	193.0
mean	1.3160621761658031
std	1.0985791184285831
min	1.0
25%	1.0
50%	1.0
75%	1.0
max	11.0

Table A.12.3 Descriptive Statistics of CC for different Functions of Test Code (Keepass)

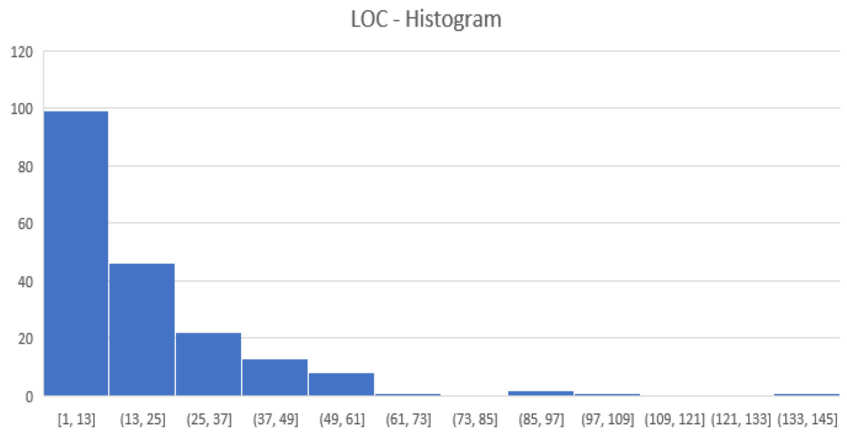
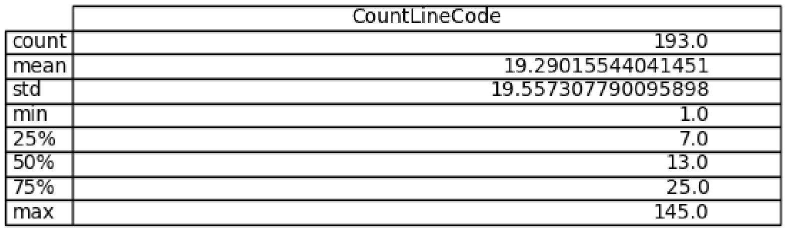


Fig A.12.4 Distribution of LOC for different Functions of Test Code (Keepass)



	CountLineCode
count	193.0
mean	19.29015544041451
std	19.557307790095898
min	1.0
25%	7.0
50%	13.0
75%	25.0
max	145.0

Table A.12.4 Descriptive Statistics of LOC for different Functions of Test Code (Keepass)

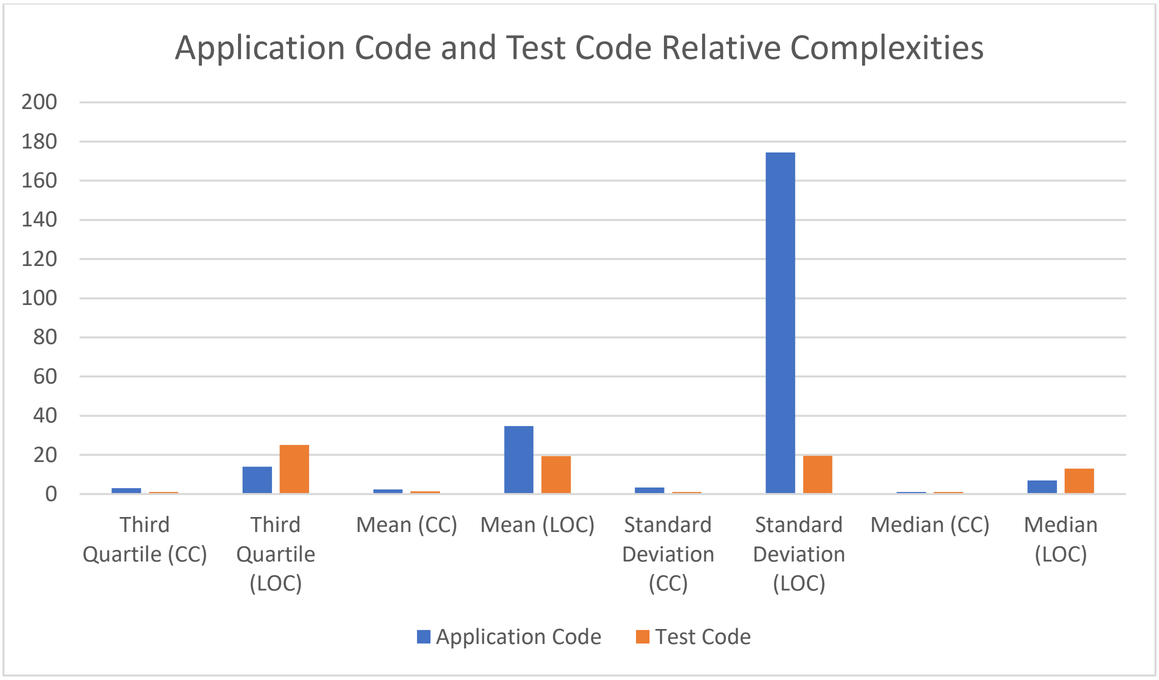


Fig A.12.5 Comparative Statistical Analysis of Application and Test Code Complexities (Keepass)

(13) Case Study [PowerShell]:

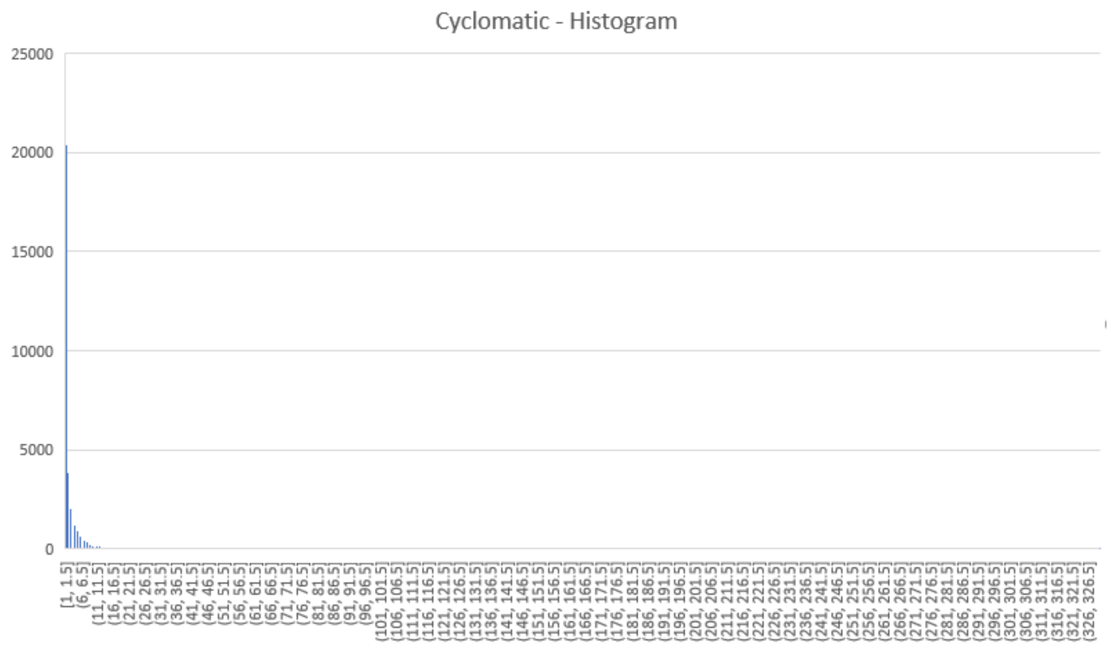


Fig A.13.1 Distribution of CC for different Functions of Application Code (Power Shell)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

	Cyclomatic
count	31308.0
mean	2.4373003705123293
std	4.538622776534671
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	330.0

Table A.13.1 Descriptive Statistics of CC for different Functions of Application Code (Power Shell)

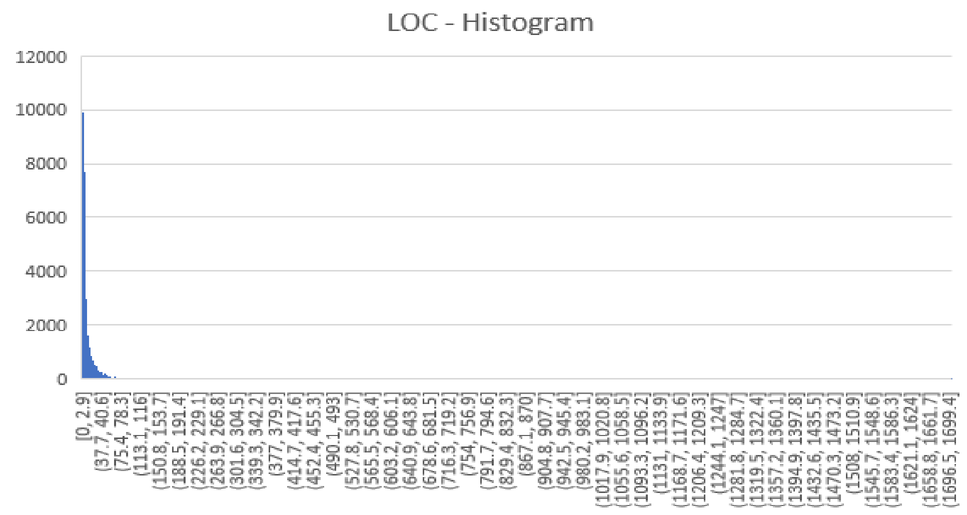


Fig A.13.2 Distribution of LOC for different Functions of Application Code (Power Shell)

	CountLineCode
count	31308.0
mean	11.804011754184234
std	26.05270905108264
min	0.0
25%	1.0
50%	4.0
75%	13.0
max	1703.0

Table A.13.2 Descriptive Statistics of LOC for different Functions of Application Code (Power Shell)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca

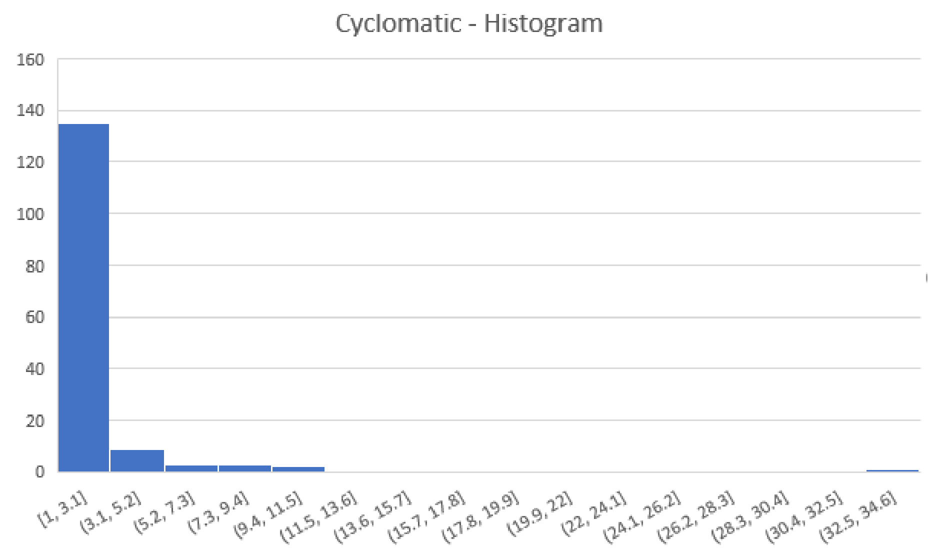
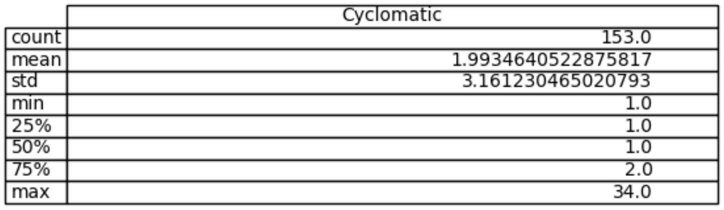


Fig A.13.3 Distribution of CC for different Functions of Test Code (Power Shell)



	Cyclomatic
count	153.0
mean	1.9934640522875817
std	3.161230465020793
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	34.0

Table A.13.3 Descriptive Statistics of CC for different Functions of Test Code (Power Shell)

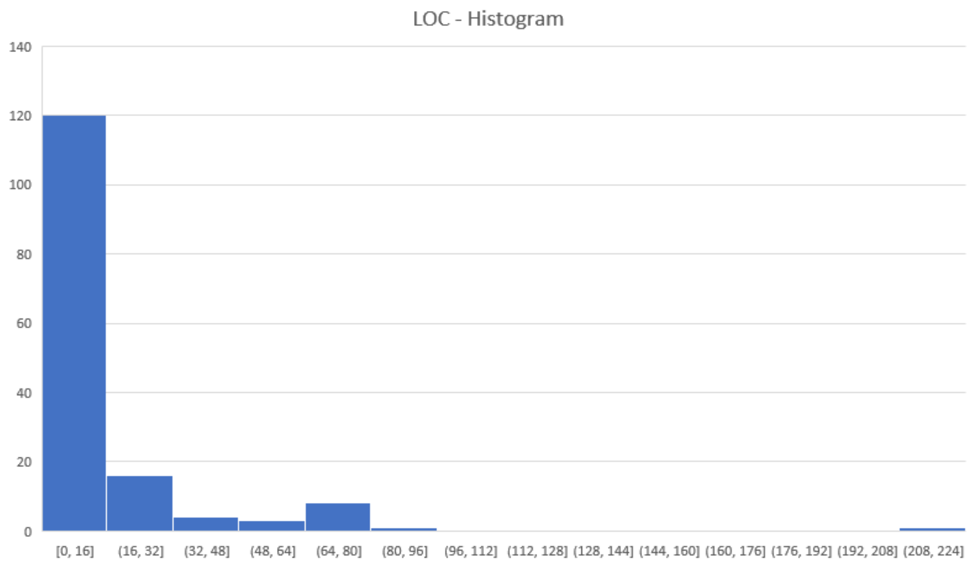


Fig A.13.4 Distribution of LOC for different Functions of Test Code (Power Shell)

	CountLineCode
count	153.0
mean	13.88888888888889
std	24.981981810990863
min	0.0
25%	0.0
50%	7.0
75%	13.0
max	219.0

Table A.13.4 Descriptive Statistics of LOC for different Functions of Test Code (Power Shell)

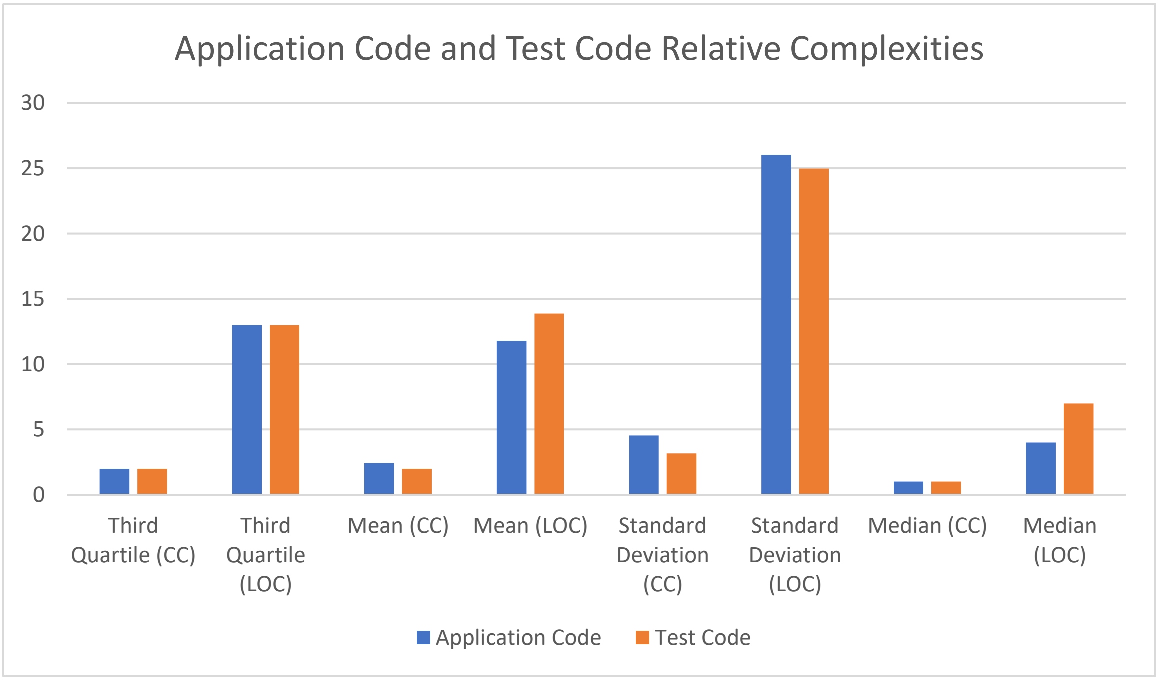


Fig A.13.5 Comparative Statistical Analysis of Application and Test Code Complexities (PowerShell)

(14) Case Study [Voldemort]:

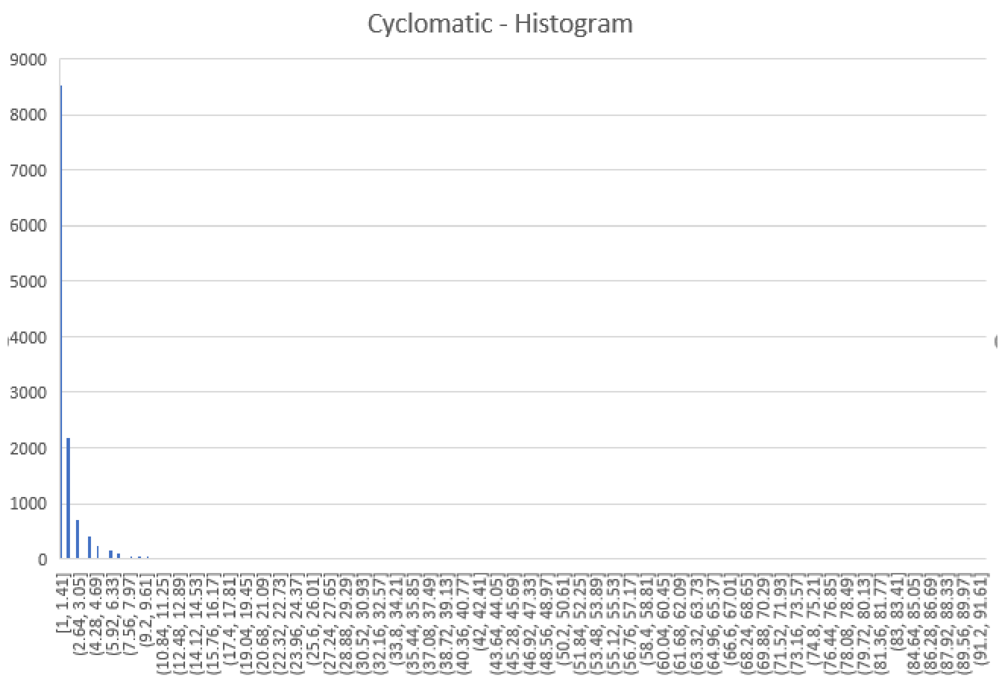


Fig A.14.1 Distribution of CC for different Functions of Application Code (Voldemort)

	Cyclomatic
count	12805.0
mean	1.936587270597423
std	2.741185482487565
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	92.0

Table A.14.1 Descriptive Statistics of CC for different Functions of Application Code (Voldemort)

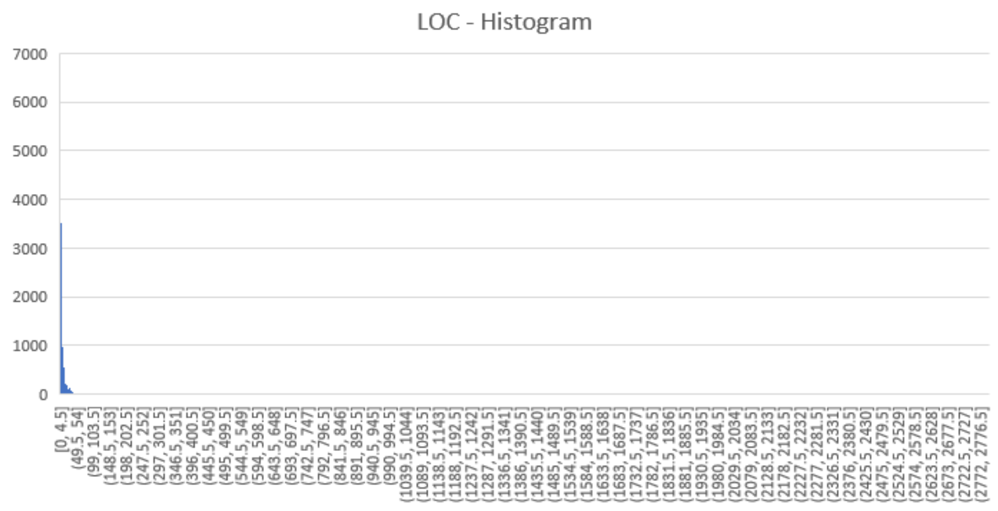


Fig A.14.2 Distribution of LOC for different Functions of Application Code (Voldemort)

	CountLineCode
count	12805.0
mean	8.790316282702069
std	30.10091586370821
min	0.0
25%	3.0
50%	4.0
75%	8.0
max	2797.0

Table A.14.2 Descriptive Statistics of LOC for different Functions of Application Code (Voldemort)

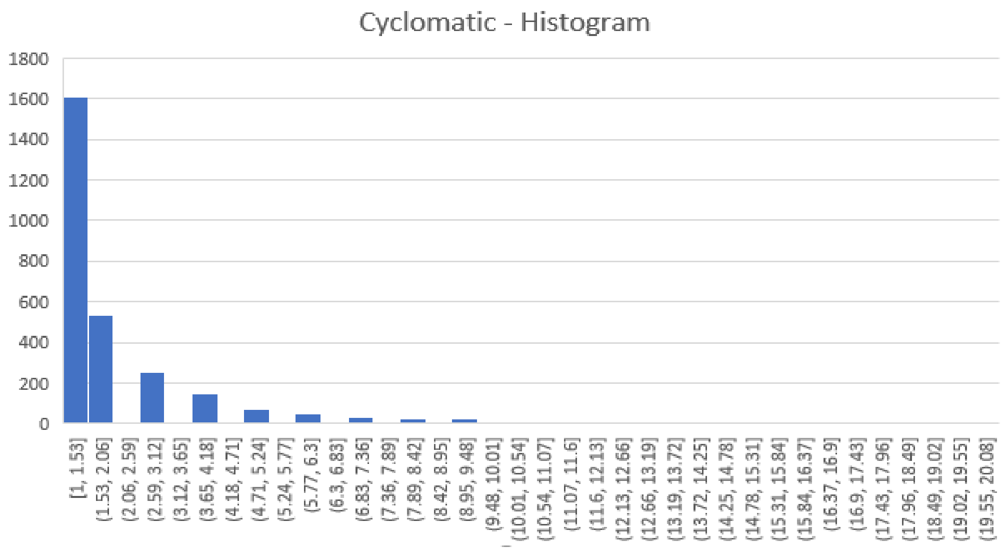
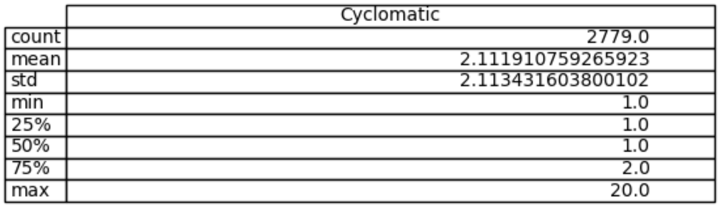


Fig A.14.3 Distribution of CC for different Functions of Test Code (Voldemort)

Navneet Kaushal
Carleton ID: 101094963
navneetkaushal@cmail.carleton.ca



	Cyclomatic
count	2779.0
mean	2.111910759265923
std	2.113431603800102
min	1.0
25%	1.0
50%	1.0
75%	2.0
max	20.0

Table A.14.3 Descriptive Statistics of CC for different Functions of Test Code (Voldemort)

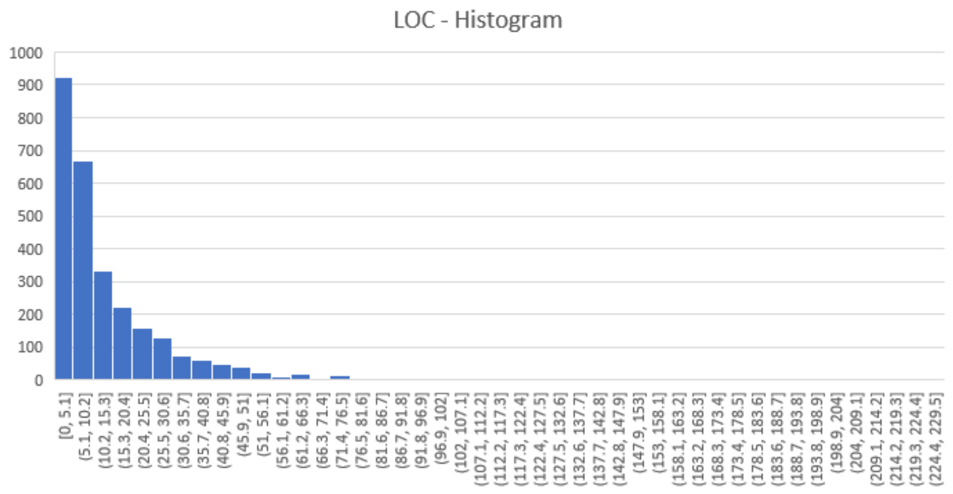
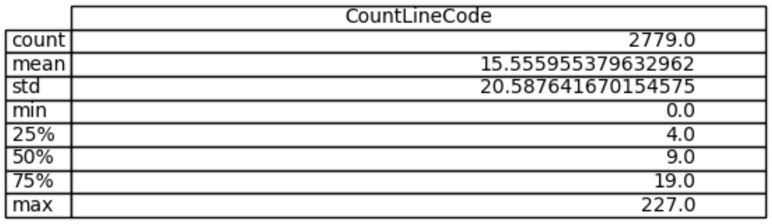


Fig A.14.4 Distribution of LOC for different Functions of Test Code (Voldemort)



	CountLineCode
count	2779.0
mean	15.555955379632962
std	20.587641670154575
min	0.0
25%	4.0
50%	9.0
75%	19.0
max	227.0

Table A.14.4 Descriptive Statistics of LOC for different Functions of Test Code (Voldemort)

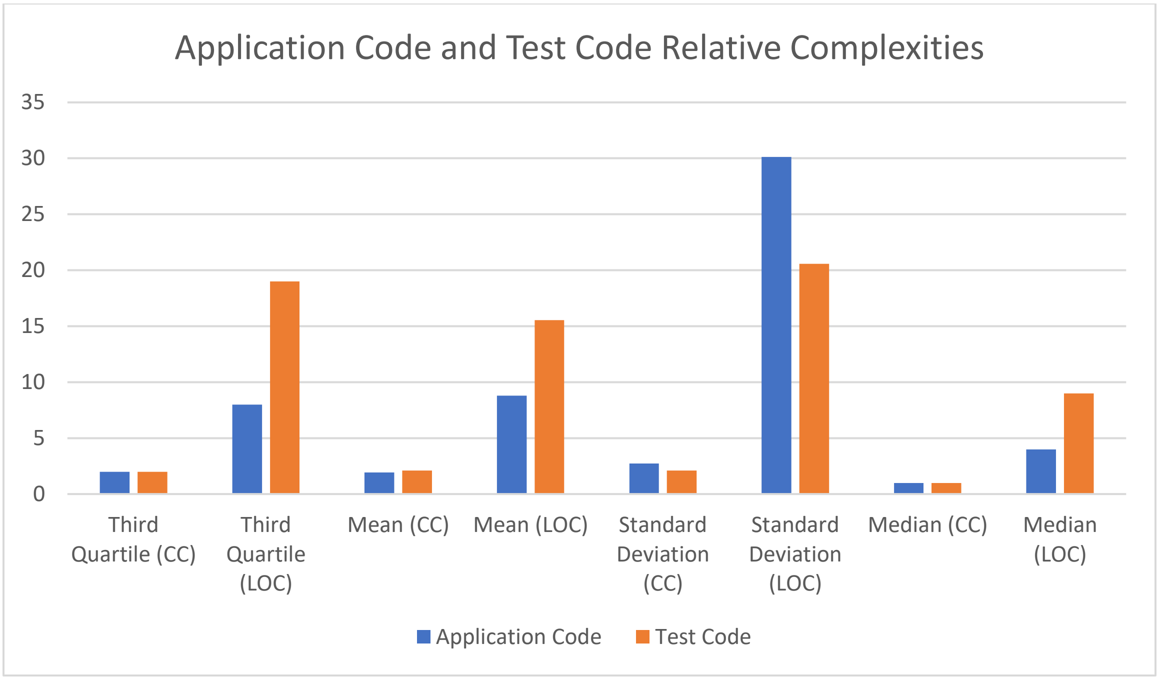


Fig A.14.5 Comparative Statistical Analysis of Application and Test Code Complexities (Voldemort)