Deriving metric thresholds for the SIG Test Code Quality Model: A benchmarking study.

Mohammed el Mochoui
m.mochoui@gmail.com

July 16, 2017, 198 pages

Academic supervisor: Jurgen Vinju, Jurgen.Vinju@cwi.nl
Daily supervisor: Pepijn van de Kamp MSc., p.vandekamp@sig.eu
Host organisation/Research group: Software Improvement Group, www.sig.eu

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Abstract

Software testing is an essential part of the software development process of which the scientific literature has shown that doing well leads to higher throughput and production code productivity. Assessing test code quality remains an open challenge. Together with the SIG, we will estimate threshold values for various test code metrics in order to get one step closer to an accurate test code quality model. We investigate in which way our code corpus should be split for estimating threshold values, which test code metrics are too complex and redundant and finally we estimate the threshold values for the metrics. We have concluded that separating production from test code, as well as Java and CSharp code is advisable because these code types are statistically very different from each other for certain metrics. In addition, we have seen that the Assert/CC, Assert/Branch and the Test:production code metrics show the least correlation with other metrics. Finally, we estimated threshold values and tested them for their ability to distinguish between well tested and poorly tested projects.

Contents

Chapter 1

Introduction

Software testing is an essential part of the software development process [1] and as much as 30-50% of the time invested in a project goes into testing [2]. Testing is done primarily to find defects in the current system, to pinpoint where these defects are located [3], and to make it possible to modify the system and add innovations without breaking parts of the system [4]. Preliminary literature shows that production code that is exercised by test code affected by test smells is more defect-prone [5]. The literature has also shown that high test code quality leads to higher throughput, the number of resolved issues per month divided by the KLOC of the system, and a higher productivity in production code [6]. Therefore, maintaining high quality test code is essential.

Assessing test code quality remains an open challenge [1]. Also, research has shown that certain source code metrics are correlated with test code effectiveness, making them a suitable candidate for a test code quality model [7]. This has led to several test code quality models being developed, for example the one described by Athanasiou [6]. The SIG has also ventured into developing a model that assesses test code quality. This research is an extension of that model, to get one step closer to a model that can be used to assess test code quality.

1.1 Problem statement

We observe assumptions made in previous research, that test and production code exhibit different characteristic properties, which leads to separation of these two code types during analysis. In addition, we also see that, for example in the SIG maintainability model [8], Java and CSharp code are described as being very similar. We want to test the assumption that include both production code and test code, and both Java code and CSharp code in the corpora that we will use to benchmark threshold metrics for the test code quality model is wrong. In the remainder of this study, we refer to including the different code types in these corpora, as we just described, as combining the different code types: Java and CSharp, and Production code and Test code. These assumptions need to be substantiated, as incorrectly combining different types of code can lead to inaccurate assessment models.
Second, the metrics used in the SIG test code quality model have been scientifically substantiated through analysis conducted by the SIG, but have not yet been calibrated. It has not been determined how the values of these metrics are interpreted, which must be done in order for the model to function as an accurate measurement tool for test code quality.
Finally, we need to provide some substantiation that the information given by these new metrics is not the same as metrics that have been researched before and are simpler to calculate, such as SLOC, CC, or the number of assert statements. When the new metrics provide the same information as these simple metrics, it is difficult to justify why the model is using these more complex, new metrics rather than the simple metrics that have already been the subject of more research.

1.2 Research questions

To address these issues, we divide the study into three parts. These three parts are briefly described below followed by a listing of the questions it answers.

Comparative analysis

In this section of the study, we will focus on the assumptions made in previous research, and will challenge these assumptions. We will focus on the following research questions:

- Can a training corpus containing both Java and CSharp projects be used to arrive at accurate thresholds for metrics measuring test quality?
- Can a training corpus containing both Production and Test code be used to arrive at accurate thresholds for metrics measuring test quality?

Correlation study

With our correlation study, we want to find out whether the new metrics make sense and whether they give us new information relative to other metrics. We calculate the pairwise linear correlation for each of the metrics. If there is strong linear correlation then we do not see a reason to extend a quality model with the extra metric since there does not seem to be any additional information captured by that metric compared to the correlating metric.

1. Is there pairwise linear correlation between the metrics used in the test code quality model?

Threshold study

In our Threshold study, we will calibrate the model to find out in what way we should interpret the values that measure the metrics to accurately asses the test code. In section 2 we introduce this model and the relevant metrics, including the Assert/Branch metrics and the production/test code ratio metrics. Based on our correlation study, we will select metrics that we think are potentially the most suitable for the model and conduct our threshold study on them. To derive the threshold metrics, we will use the Alves [9] and Baggen [10] methodologies, described in previous literature. We will introduce these methodologies in section 2.

These metrics can be analyzed at the unit, file, and system levels, where unit is defined as to a function or method in Java or CSharp systems [8]. For each metric, we need to consider whether the metric value at the file level is equivalent to summing the metric values of all the units it contains, or whether they should be measured individually at the file level. The same applies at the system level. An example would be the number of lines of code in a unit, file, or overall system.

We answer the following questions about the selected metrics:

1. What is the risk profile of these metrics at the file level?
    - What percentiles of code do we use for estimating our risk profile?
2. What are the system-wide threshold values of these metrics, to arrive at a 5-star score?
3. Which metrics are not compatible with the Alves and Baggen methods and require a deviation from them?

1.3 Research method

Comparative study

To find out what the test code qualities look like, we will perform a statistical analysis on a dataset supplied from the proprietary system corpus of the SIG. We also take a look at the distributions of these systems. We do this at the unit, file and system level. Next, we use these statistical values and distributions to compare the code types, in order to conclude in what ways these types of code differ and if it is advisable to combine the code types to derive metric thresholds or to separate them.

Correlation study

To find out if our test code metrics provide us with different information than the simple metrics such as SLOC, CC and the number of assert statements, we will calculate the Pearson correlation between these metrics. We chose the Pearson method because it describes the linear correlation between two variables [11]. This linear correlation describes the extent in which one variable can be used to estimate the value

of another variable. High predictability indicates that the two variables provide us with largely the same information and that one of them is redundant. We will then interpret these correlation values after which we can conclude which metrics are redundant and provide us with the same information.

We interpret variables with high correlation as redundant because we believe that this high predictability between these two variables indicates a high change of the existence of an underlying factor that explains both metrics, or that one explains the other. To definitively rule this out, we would need to examine the metrics with high correlation, to see if there is also a plausible causation that causes this high correlation. This is beyond the scope of this study.

Threshold study

To determine the threshold values, and threshold percentiles, we will focus on two research methods that have been established in previous studies. Using the threshold derivation method described by Alves [9], we will create risk profiles at the unit or file level. On the system level we use the Baggen method [10] to estimate a star score.

1.4 Contributions

We divide the contributions of our research in the three main parts of our research. Our research makes the following contributions:

Comparative study

1. We offer insights into what production and test code looks like using statistics.
2. We test the assumptions in previous studies where the corpora consisted of either production code or test code, but consisted of both Java code and CSharp code.
3. We offer a statistics based answer whether to include both production code and test code, and both Java code and CSharp code in the corpora that we use to benchmark threshold metrics for the test code quality model.

Correlation study

1. We introduce modifications to the SIG test code quality model based on the linear correlations that the metrics exhibit with metrics that are easier to compute.

Threshold study

1. We show the cumulative distribution of the metrics on unit and file level and choose the appropriate thresholds at which to estimate the metric values.
2. We provide threshold values and risk profiles for the selected metrics, which can be used in the test code quality model to assess test code. The selected metrics will be mentioned in the introduction of chapter 8. The selection of the metrics is based on chapters 6 and 7.

Chapter 2

Background

2.1 Terminology

In this section we will explain the terminology the will be used in this research. These will mainly be software metrics that we are researching.

2.1.1 Metrics

LOC

In this study, we use the LOC metric as defined in the SIG's maintainability model [8]. We define the LOC metric as all lines of code that are not blank or comment lines.

Unit

We also use the definition from the SIG's maintainability model for the unit metric [8]. A unit is equivalent to a function or method in Java or CSharp systems.

Cyclomatic complexity (CC)

The Cyclomatic Complexity describes the complexity of a unit. We calculate it by counting the number of control flow statements and adding 1 to this. For Java and CSharp, the control flow statements include: if, case, ?, ??, &&, ||, while, for, foreach and catch [12].

Branch point

A branch point is one of the statements mentioned above and is thus a synonym for control flow statements. Thus, the number of branch points in a unit is equal to the number of CC minus 1.

Assert

An assertion is defined in the Java docs as an expression by which we can test assumptions about our code [13]. By examining the results of the tool we used for this research, we found that the tool does not distinguish between assert statements from Java and CSharp itself, and assert statements defined in 3rd party libraries.

2.1.2 SIG Tools

SAT

The SAT, Software Analysis Tool, is the tool developed by the SIG and used for analyzing systems at the unit, file, and system level. This is the tool used to generate the data for this study.

SAW

The SAW, Software Analytics Warehouse, is the database in which all analysis data generated by the SAT is stored. The corpus used in this study was queried from this database.

2.2 Statistics

2.2.1 Skewness & Kurtosis

To describe our dataset and compare the different categories to find out which types of code can be analyzed together and which should be taken independently we will look at the distribution of the data, among other things. We will use the Kurtosis and Skewness metrics to compare the distributions, in addition to the Lorenz curves and the Gini coofiecient. We are mainly interested in the Skewness and Kurtosis because these metrics describe clear properties of the distributions, which can be easily compared between different datasets.

Skewness

The Skewness is a metric that defines the symmetry of a distribution. This metric can be positive, negative or 0.

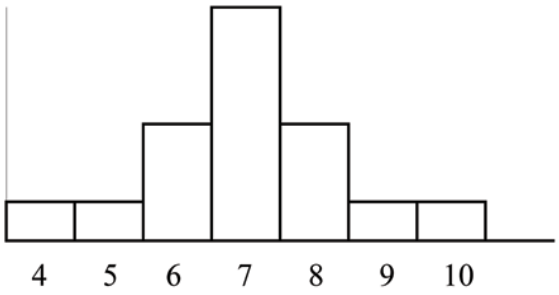

Figure 2.1: An example of a perfect symmetric distribution. [14]

In figure 2.1 we see an example of a perfectly symmetric distribution. It is characterized by having a mean and median that are equal to each other [14].

In the case of a non-symmetric distribution, we experience one of the two following situations.

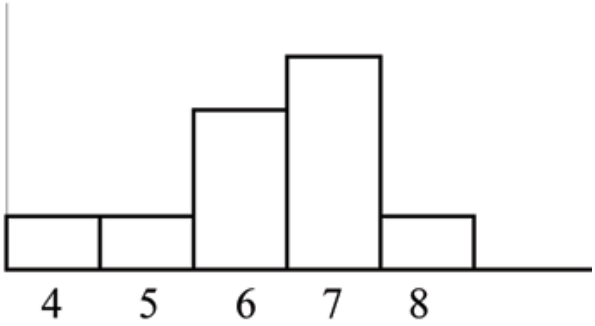

Figure 2.2: A left skewed asymmetrical distribution. [14]

In figure 2.2 we see an example of an asymmetrical left skewed distribution. We speak of a Skewness to the left if the mean is less than the median [14].

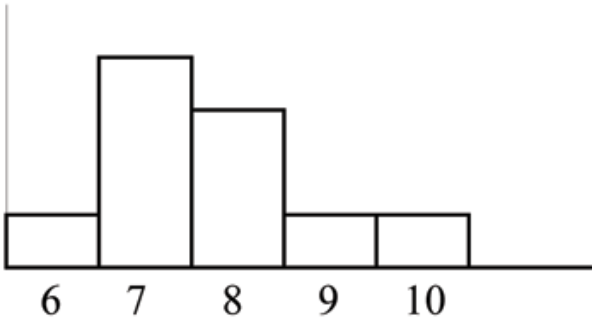

Figure 2.3: A right skewed asymmetrical distribution. [14]

In figure 2.3 we see an example of an asymmetrical right skewed distribution. We speak of a Skewness to the right if the mean is greater than the median [14].

The author of the article summarizes this as: "generally if the distribution of data is skewed to the left, the mean is less than the median, which is often less than the mode. If the distribution of data is skewed to the right, the mode is often less than the median, which is less than the mean." [14].

Kurtosis

The Kurtosis is a value that describes the tailedness of a distribution. The interpretation of this metric is defined as follows: "its only unambiguous interpretation is in terms of tail extremity; i.e., either existing outliers (for the sample Kurtosis) or propensity to produce outliers (for the Kurtosis of a probability distribution)" [15].

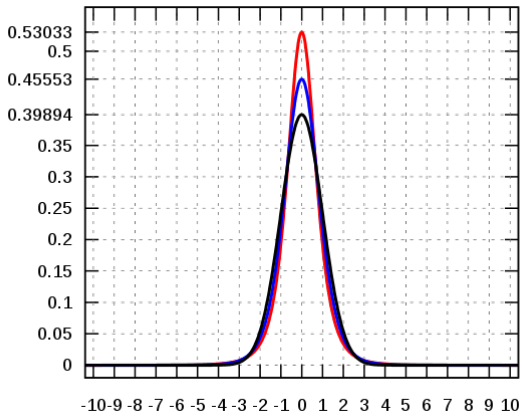

Figure 2.4: Examples of distributions with their Kurtosises. (Red: infinity, Blue: 2, Black: 0) [16]

Distributions can be divided into three different categories when it comes to them Kurtosis, depending on the excess Kurtosis. This is the Kurtosis from which 3 is subtracted.

Mesokurtic

A Mesokurtic distribution has an excess Kurtosis of 0, and thus a Kurtosis value of 3. An example of a distribution that falls under the Mesokurtic category is the normal distribution.

Leptokurtic

A Leptokurtic distribution has a positive excess Kurtosis which means that this distribution produces more outliers. An example of a distribution that falls under the Leptokurtic category is the exponential distribution.

platykurtic

A Platykurtic distribution has a negative excess Kurtosis and has thinner tails, implying that this distribution produces fewer outliers. An example of a distribution that falls under the Platykurtic category is the Bernoulli distribution. ‘

2.3 Threshold benchmarking

2.3.1 Alves method

In this section, we examine a methodology for determining thresholds using benchmarking data as presented by Alves [9]. This is also the methodology used by SIG to determine metrics thresholds for their models. The methodology is characterized by the following three features:

- Data-driven: It uses real world code corpora.
- Robust: The outcome is not influenced by outliers.
- Pragmatic: The method should be easy to repeat.

The Alves paper [9] states that previous research has derived metric thresholds for test code quality models that were primarily based on:

- Experience
- Metric analysis
- Error models
- Cluster Techniques

The method advocated by this research is a benchmark-based method to derive thresholds. It consists of the following steps:

1. metrics extraction: Metric values are calculated over the code. A weight is assigned to this metric that is based on the number of LOC. This weight will be used for normalizing.
2. weight ratio calculation: The weight ratio is calculated for each method.
3. The weights of all units are aggregated, The result of this step is a weighted histogram.
4. weight ratio aggregation: A matrix is derived by sorting the metric values and taking the maximal value as a representation for a certain percent of the code.
5. thresholds derivation: A threshold value is determined by picking a percentage and reading from this graph what metric corresponds to it.

2.3.2 Baggen method

The Baggen paper [10] provides a high-level overview of SIG's method for code analysis and quality consulting focused on software maintainability. This includes calibrating metrics scores over a 5-star system at the system level. We do this by sorting the systems from a low to high score per metric, and then dividing them into the following percentiles: 5, 35, 65, 95 and 100. This ensures that systems that score 5 stars for a given metric are among the 5% highest scoring systems for that metric. Determining thresholds in this way has a number of advantages, including [10]:

- It is based on data, and therefore objective.
- It can nearly be automated, which allows for easy updates on the thresholds of a model.

The SIG uses this method to benchmark their models yearly.

Chapter 3

Related work

3.1 Athanasiou's test code quality model

In Athanasiou's study [6], a model for test code quality is proposed. This model consists of the following metrics:

- Code Coverage
- Assert/CC Ratio
- Assert density (The amount of assert statements divided by the total Test-LOC)
- Directness (The percentage of production code that is directly called by the test suite)
- Maintainability of test code

These metrics map to the subcategories completeness, effectiveness and maintainability. The aggregation of the properties per sub-characteristic is performed by acquiring the average.

This model is benchmarked using the Baggen method [10]. On a system level the metric score is expressed on a 5 star system based on the (5, 35, 65, 95) percentiles. On metric levels, the risk profiles are derived using the Alves method [9], and are sorted into the categories low, moderate, high and very high. The boundaries for these risk groups are defined as the 70th, 80th and 90th percentiles.

3.2 SIG Test code quality model

SIG has also designed a model by which test code quality can be assessed. A Test Quality model with 3 base characteristics and 6 system properties was defined. The model is defined to address the following problems related to software testing:

- Changing not well-tested code risks the introduction of bugs.
- Having knowledge of which code parts are not well tested helps to localize the risk.
- Having knowledge of how not-well tested the code is helps assessing the amount of risk.

In table 3.1 we can see the conclusions drawn during the studies done by the SIG to design the model.

Metric	Conclusion
Code coverage	Higher code coverage results in less defects per line of code. Higher code coverage results in a faster defect resolution
Mutation coverage	Mutation coverage is a good indicator of test effectiveness.
Assert/McCabe ratio	The ratio of assert and CC is a good indicator of test effectiveness Higher assert/McCabe ratio results in a higher throughput (solved issues/KLOC) and productivity (solved issues/per dev). Software consultants have indicated that they do not agree with measuring the assert statements against the CC, since this would mean that getters and setters have to be tested as well.
Maintainability of test code	Maintainability of test code has no relation with the issue reslution time
Test smells	Code smells in the test code indicate test code with a high likelihood to contain bugs. Code smells in the test code do not indicate a high likelihood for bugs in production code.
Test code / Production code Ratio	The test:production code ratio metric can be used as a predictive metric for code coverage. This metric does not call to action and has no straightforward solution.
Asserts / Branch points Ratio	As a solution for the problem that software consultants dislike the use of the Assert/CC metric, the amount of branch points can be used instead.
Proportions of test case types	This is an industry wide accepted concept, that has no thresholds yet.·

Table 3.1: Summary of the results of the studies done leading up to the SIG test code quality metric.

The model consists of the following metrics:

- Test:Production code Ratio
- Asserts/Branch points Ratio
- Code Coverage
- Proportions of test case types
- Mutation coverage

The model consists of the following three sub-characteristics:

- Completeness. (Test:production ratio, code coverage)
- Effectiveness. (Assert:branch point ratio, Mutation coverage)
- Efficiency. (Test type proportion)

All these metrics are measured on a 5 star scale, that SIG also used in their other models [8]. The model is not yet in production and the metrics have yet to be calibrated through a benchmarking study. In Section 8, we will take a look at some of these metrics and benchmark them to find out the associated thresholds.

Chapter 4

Dataset description

4.1 Introduction

In this section, we would like to introduce the reader to the corpus used for this study, and to explain the creation of this corpus. In addition, we try to find the threats to validity that are introduced by the use of this corpus for this study. We will do this using randomly selected samples in which the data points are anonymized. We will also motivate in what way we filtered the data points. The questions this section answers can be summarized as:

- How was this corpus established and prepared?
- What are the frequencies of each category of code?
- Are there code categories that are underrepresented?
- What potential threats to validity can we identify from this analysis?

Answering these questions lays the foundation for this research and provides the reader with information about the creation of the dataset that will be used in all experiments. Finally, this section serves as a basis for the following sections in which we delve deeper and attempt to answer the research questions. In this section, we anticipate on this and use the insights we gain through this experiment to adjust our expectations of the result. The corpus we use consists of Java and CSharp units, files and systems measured by the SAT, and stored and queried from the SAW.
Because of the linear interpolation used in section 6, all metric values in this study are presented as floats. Integer metrics might be presented as halves, which is technically incorrect, but does not impact the end result of this study because assessing real world projects with integer metric values works the same as ceiling the result to the nearest integer. This float representation will be used throughout this research.

4.2 Method

We used Python [17] and Pandas [18] to calculate the frequency table which we present in Table 4.1. We also used Pandas to sample from our dataset to present the sample data points in Tables 4.2, 4.3 and 4.4. We then used the frequency table 4.1 to calculate the ratios between the corresponding code categories.
By observing the data and looking at choices made in preparing the dataset we identified potential threats to validity. We used the ratios between the corresponding code categories in Table 4.5 and the absolute numbers in Table 4.1 to see if any categories were underrepresented.

4.3 Dataset preparation

4.3.1 Querying

The dataset is sourced from the SIG and stored in the SIG data warehouse. The SIG data warehouse is a database which stores the metric values on a unit, file and system level. We have queried the data on a unit, file and system level from the data warehouse with a Python script using the PyMongo database. The criteria set for querying the data are:

- Language: Java or CSharp
- Type: Production or Test
- Snapshot date: Greater than 01-09-2020.

We select only the snapshots taken after introducing the test code quality model where the number of branch points and the number of direct asserts are also measured.

4.3.2 Filtering

We stored this data in three different CSV files, for unit, file and system data separately. We read this data into a Python Pandas DataFrame in the Jupyter Lab environment. Using these data frames, we were able to inspect the data for strange values.

General filtering

We have found a number of systems with an incorrect snapshot date that is in the future. We filtered these out of the dataset, as we do not know if the rest of the metrics are also incorrect. We additionally filtered out all units, files and systems that do not have both production and test SLOC. This ensures that we analyze for the same systems and code at the unit, file and system level.

Unit level

At the unit level, we filtered out the default methods. The SAT defines default unit as all code that is not in a unit but is in a class. Often these are the instance variables. These are measured by the SAT as units with a SLOC value of N/A. We filtered these out because we interpreted the default units to not be units, but code outside the units.

File level

At the file level, we converted all N/A values for the direct assert metric to 0. At the implementation level, it was decided that the SAT assigns an N/A value to all files for which no corresponding test file was found. This essentially means that the file was not tested at all, or tested by not adhering to the naming conventions of test files. We therefore interpret the absence of the corresponding test file as 0 direct asserts.

System level

Also at the system level, we converted the N/A values for the direct assert metric to 0. This is according to the same reasoning as for files.

4.4 Frequencies

	Units	Files	Systems
Total	10525855	1637630	1001
Production	9130799	1286979	1001
Test	1395056	350651	1001
Java	5184720	783232	517
CSharp	5341135	854398	493
Java production	4420262	598932	517
Java test	764458	184300	517
CSharp production	4710537	688047	493
CSharp test	630598	166351	493

Table 4.1: Amount of data points per category in the dataset.

Table 4.1 shows the numbers by category in our dataset. At a unit level, we see that 88% of our units are production units. Deriving metric thresholds on a dataset that combines production and test code using this dataset introduces problems if we find out that production and test code differ significantly. The results would be largely influenced by production code, and only a small part by test code.
We see that we have about 52% more Java systems than CSharp systems in our test code, despite the fact that Java and CSharp unit and file counts differ little from each other. In fact, we see that we have more CSharp files than Java files. CSharp units and files are distributed over a smaller number of systems than Java units and files.

4.5 Sample data points

4.5.1 Unit level

	type	lang	SLOC	CC	branchpoints
0	production	Java	14.0	1.0	0.0
1	production	Java	3.0	1.0	0.0
2	test	Java	3.0	1.0	0.0
3	production	Java	3.0	1.0	0.0
4	production	CSharp	1.0	1.0	0.0

Table 4.2: A random sample of 5 points of the unit level data.

In Table 4.2, we see 5 example data points from our unit-level dataset. At the unit level, we have 3 metrics: SLOC, CC and branch points. In addition, we also included the language and type of each unit in our dataset. These are determined by the SAT while analyzing the Java projects.

4.5.2 File level

	type	lang	SLOC	CC	branchpoints	asserts	direct asserts
0	test	Java	469.0	31.0	0.0	27.0	0.0
1	test	CSharp	67.0	9.0	2.0	0.0	0.0
2	production	Java	33.0	2.0	0.0	0.0	0.0
3	production	CSharp	24.0	9.0	0.0	0.0	0.0
4	production	Java	49.0	14.0	9.0	0.0	0.0

Table 4.3: A random sample of 5 points of the file level data.

At the file level, we see the same metrics recurring as at the unit level. In addition, we see two new metrics that are not measured at the unit level. At the file level, the number of assert statements per file is also stored. In addition, for each file, the number of direct asserts measured is stored.

4.5.3 System level

	type	lang	SLOC	CC	branchpoints	asserts	direct asserts
0	production	Java	125249.0	26605.0	13208.0	0.0	0.0
1	test	CSharp	810.0	80.0	2.0	18.0	0.0
2	production	CSharp	66664.0	14760.0	4363.0	550.0	4707.0
3	production	Java	5671.0	1087.0	361.0	0.0	0.0
4	test	Java	3435.0	436.0	18.0	249.0	0.0

Table 4.4: A random sample of 5 points of the system level data.

Systems level metrics are also measured independently by the SAT. We chose to aggregate the file level data to arrive at the system level data. This aggregation is done by summing the metrics. Since all code in a system resides in a file, aggregation by summation is correct for the above metrics. This is not true for aggregation from unit level to file level, as not all code in a file is in units. We will see examples of this in section 5. Examples of code pieces that reside in files but not in a unit are instance variables and import statements.
The decision to aggregate the metrics rather than use the SAW data at the system level fell due to the lack of a documentation regarding querying the data from the SAW, making this a problematic process. In addition, implementation decisions in the SAT, such as the complexity of excluding default units at the file level but including them at the system level, leads to the SAW system level data is not a correct summation from file level to system level. Since we could not find any motivation for these implementation choices, which we also could not find in previous literature, we adhere to the aggregation of file level to system level by summation.

4.6 Analysis

In this section, using the data delivered above, we attempt to answer the questions posed in the introduction. In particular, we will focus on the potential under-representation of a code category and the threats to validity that we can identify from the dataset or its creation.

4.6.1 Under-representation

By studying Table 4.1, we can answer the question of whether there are code categories that are under-represented. We see that we have a lot less data points on all test categories. The ratios between the corresponding code categories are listed in Table 4.5.

	Unit	File	System
Production : Test	6.55	3.67	1
Java : CSharp	0.97	0.92	1.05
Java production : Java test	5.78	3.25	1.05
CSharp production : CSharp test	7.47	4.14	1.05

Table 4.5: Ratios between corresponding code categories.

In table 4.5 we see the ratio's between the different code categories. On a system level we see ratio's between the code categories close to 1. This is because we have only queried systems from the SAW that contain both production and test code, and these ratios are based on the number of data points, in this case systems. On a unit and file we see higher ratio's between all production and test categories. If the ratio between production and test code is too large, production code will be dominant in the threshold calculations. In the event that production and test code individually provide very different threshold values for the metrics, the combination of these two categories will cause the combined threshold values to be much closer to production code than test code, making it inaccurate for evaluating test code.

	Units	Files	Systems
Total	71302778.0	126547634.0	126547634.0
Production	59261827.0	86109590.0	86109590.0
Test	12040951.0	40438044.0	40438044.0
Java	33531240.0	60827283.0	60827283.0
CSharp	37771538.0	65720351.0	65720351.0
Java production	28439295.0	41777028.0	41777028.0
Java test	5091945.0	19050255.0	19050255.0
CSharp production	30822532.0	44332562.0	44332562.0
CSharp test	6949006.0	21387789.0	21387789.0

Table 4.6: Amount of SLOC per category in the dataset.

In the Alves and Baggen method the data points are normalized by their SLOC [9, 10] to derive accurate threshold metrics. Table 4.6 displays the distribution of the SLOC among the different categories. Here we see that almost all ratio values on a unit and file level have gone down compared to the ratios in table 4.5, which reduces their impact on the combined threshold values. Nevertheless, the values at unit and file level are still high and undesirable for the combined threshold values, as the result is more influenced by the bigger variable.

	Unit	Level	System
Production : Test	4.92	2.13	2.13
Java : CSharp	0.89	0.93	0.93
Java production : Java test	5.59	2.19	2.19
CSharp production : CSharp test	4.44	2.07	2.07

Table 4.7: Ratios between corresponding code categories based on SLOC.

4.6.2 Potential threats to validity

Production-test frequency ratio

The large difference in the number of production and test units and files is a potential threat to validity, depending on whether production and test code differ significantly from each other. This would mean that production code exerts a great deal of influence on the outcome results when production and test code are analyzed in combination to determine threshold values.

Missing data

Entering 0 values with the direct assert metric in case the corresponding test file is missing is a choice that carries a threat to validity. It can affect the final results since the 0 values do not properly indicate whether testing was done but there are no direct asserts or no tests were written at all. By entering 0 values in the direct asserts metric, we give the impression that the corresponding test files were found, but that they do not contain assert statements.

SAT implementation of systems

Finally, systems occur in our data set that have a data point for CSharp and Java. This is then a system that consists of both CSharp and Java. There is no minimum threshold defined for the size of a system. Thus, a system that consists largely of CSharp code and a single file with 5 lines of code Java, will be measured as two independent systems, one for CSharp and one for Java with 5 lines of code. This gives the impression that there is a Java system with 5 lines of code, where in reality this is a CSharp project with a small piece of Java code. In table 4.8 we see the 5 smallest systems in our dataset, that illustrate this problem.

	SLOC	CC	Branchpoints	asserts	Direct asserts
0	20.0	4.0	0.0	0.0	0.0
1	20.0	4.0	0.0	0.0	0.0
2	20.0	4.0	0.0	0.0	0.0
3	21.0	4.0	0.0	0.0	0.0
4	21.0	4.0	0.0	0.0	0.0

Table 4.8: The 5 smallest systems in the dataset.

4.7 Conclusion

In Section 4.3, we discussed the steps we took to arrive at the dataset that will be used for the Analysis during this study. We first queried the appropriate data from the SIG's database and then filtered these values to remove unwanted data points correct undesired values.
We divided our dataset into code categories and calculated their frequencies. These can be found in Table 4.1.
Using this frequency table, we calculated the frequency ratios between corresponding code categories. These frequency ratios are listed in 4.5. From this table we are able conclude that there is a large difference between production and test code and that production code is dominant when the data set is used for determining thresholds, without separating production and test code.
By analyzing the above data, we identified the following threats to validity:

- In the case of a difference between production and test code, combined analysis of these two code categories will yield results where production code exerts a more dominant influence. This threatens the validity of the estimated threshold metrics.
- By filling in 0 values for missing values in the direct assert metric, the information about the existence of the associated test file is lost. This threatens the validity of the estimated threshold

metrics because the original data has been altered without proof of the correctness of this alteration. The modifications of the original data influence the threshold metric estimations.

- Handling missing production or test code on a system level by filling in the values with 0 causes the thresholds to be lower than they really are. This influences the threshold metrics the same way the previous threat does.
- The way the SAT implements a system gives the impression that certain systems are tiny, when in fact they are only part of the overall system. This also affects the results of statistical calculations, as more system data points with small values are included in the data set. This is a threat to the validity of the conclusions drawn on a system level by interpreting the term system differently from other literature [19]. Conclusions drawn in this study on a system level might not be applicable or comparable to conclusions drawn in other literature, because the terminology is different.

Chapter 5

Exploratory data analysis

5.1 Introduction

In this section we will describe our dataset. We have made the choice to describe all the data together as we do not yet see any reason to split the data. In the previous literature [8, 19] we have seen no demonstrable differences between test and production code, or between Java and CSharp code. We describe the dataset so that the reader develops an understanding of the data used for the analysis in this study. We will look at how data points translate to code with focus on edge cases. The questions this section answers can be summarized as:

- What does the frequency distribution of our dataset look like?
- Is the mean representative for this dataset?
- How do interesting data points translate into code?

The distributions of the datasets are important factors in determining the threshold values according to the Alves [9] and Baggen method [10]. In addition, in the next section we will find out whether CSharp, Java, production and test code can be combined when determining threshold values. For this, we want to know if the average can be used as a starting point for comparing these code categories. There again, the distribution of datasets plays a major role. Finally, we also want to be able to translate these data points into real code, to paint a picture towards what these values mean in code.
We will qualitatively describe the distributions using histograms in which we can visually see the distribution. Quantitatively, we will look at the Skewness and kurtotis values, which describe the asymmetry and tailedness of the distribution. Using the Kurtosis and Skewness, we can conclude how representative the mean is of the data points in the dataset.

5.2 Research method

We will answer these questions by generating tables that display descriptive statistics (quantiles, mean, min, max, Skewness and kurtosis) for the data. The quantile values give us the maximum value for each metric at this quantile of data points. We use the Skewness and Kurtosis metrics to describe the the shape of the distribution. In addition, we also present histograms of the data with which we visualize this distribution.
By calculating and presenting descriptive statistical metrics in the table and visualizing the frequency distribution through histograms we describe the distribution. With this distribution we can conclude whether the average is a representative value. We then use these histograms and tables to identify interesting values, after which we can look at how these translate into code.

We chose to analyze the entire dataset and we do not split the dataset by code category. This is because we have not yet confirmed nor disproved the assumption that production, test, Java and CSharp code differ from each other. Since we do not yet know if these categories differ from each other, and how they differ from each other, we do not split the dataset. In Chapter 6, we will look at whether and how the categories differ from each other.

5.3 Hypothesis

Hypothesis 1. *We expect skewed long tail distributions on a unit and system level.*
In previous research we have seen that at the unit and system level these distributions have been observed [19]. We expect a high Skewness value and that small values dominate the dataset.

Hypothesis 2. *We expect the mean to not be representative for this dataset.*
We expect this based on the dominance of the small values and the influence that the large outliers have on the mean in the skewed long tail distributions [19].

5.4 Results

5.4.1 Unit level

Tables

	min	5%	25%	50%	75%	95%	max	mean	skew	kurt
SLOC	0.0	0.0	1.0	3.0	7.0	24.0	15549.0	6.77	104.44	74632.49
CC	1.0	1.0	1.0	1.0	1.0	5.0	1084.0	1.73	37.78	5294.09
Branchpoints	0.0	0.0	0.0	0.0	0.0	4.0	1083.0	0.73	37.78	5294.09

Table 5.1: Descriptive statistical metrics for Units in the dataset.

SLOC	Frequency
3.0	2642094
0.0	1867374
1.0	1530671
4.0	656239
5.0	402449

CC	Frequency
1.0	8276125
2.0	918243
3.0	479346
4.0	272342
5.0	163333

Branchpoints	Frequency
0.0	8276125
1.0	918243
2.0	479346
3.0	272342
4.0	163333

Table 5.2: Frequency tables for unit level metrics.

Plots

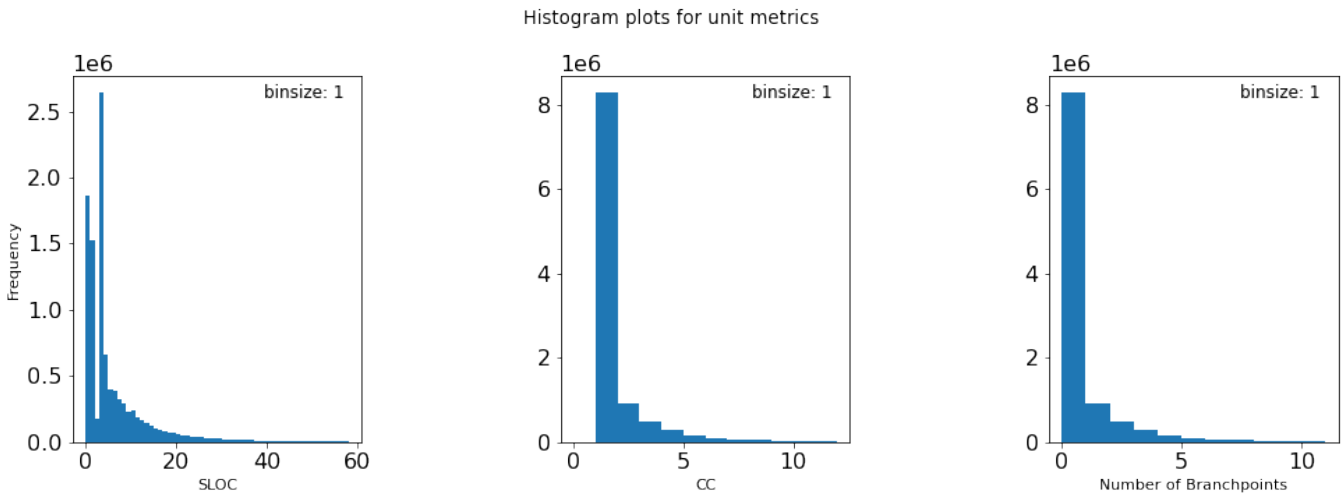

Figure 5.1: Histogram of unit level metrics.

5.4.2 File level

Tables

	min	5%	25%	50%	75%	95%	max	mean	skew	kurt
SLOC	0.0	8.0	16.0	34.0	76.0	265.0	68502.0	77.27	52.99	10378.40
CC	0.0	1.0	2.0	5.0	11.0	41.0	5822.0	12.09	30.80	2518.59
Branchpoints	0.0	0.0	0.0	0.0	3.0	22.0	4349.0	4.99	30.55	2179.50
asserts	0.0	0.0	0.0	0.0	0.0	11.0	5430.0	2.39	81.02	17771.86
Direct asserts	0.0	0.0	0.0	0.0	0.0	4.0	2766.0	1.06	51.34	8385.31

Table 5.3: Descriptive statistical metrics for Files in the dataset.

SLOC	Frequency
9.0	41175
10.0	40784
12.0	39732
11.0	38860
8.0	38408

CC	Frequency
2.0	355492
3.0	169787
4.0	146923
5.0	111011
6.0	94835

Branchpoints	Frequency
0.0	995624
1.0	129851
2.0	92648
3.0	62379
4.0	49159

Asserts	Frequency
0.0	1293361
1.0	73990
2.0	48915
3.0	30652
4.0	26988

Direct asserts	Frequency
0.0	1518875
2.0	13044
4.0	10739
3.0	10545
5.0	8151

Table 5.4: Frequency tables for File level metrics.

Plots

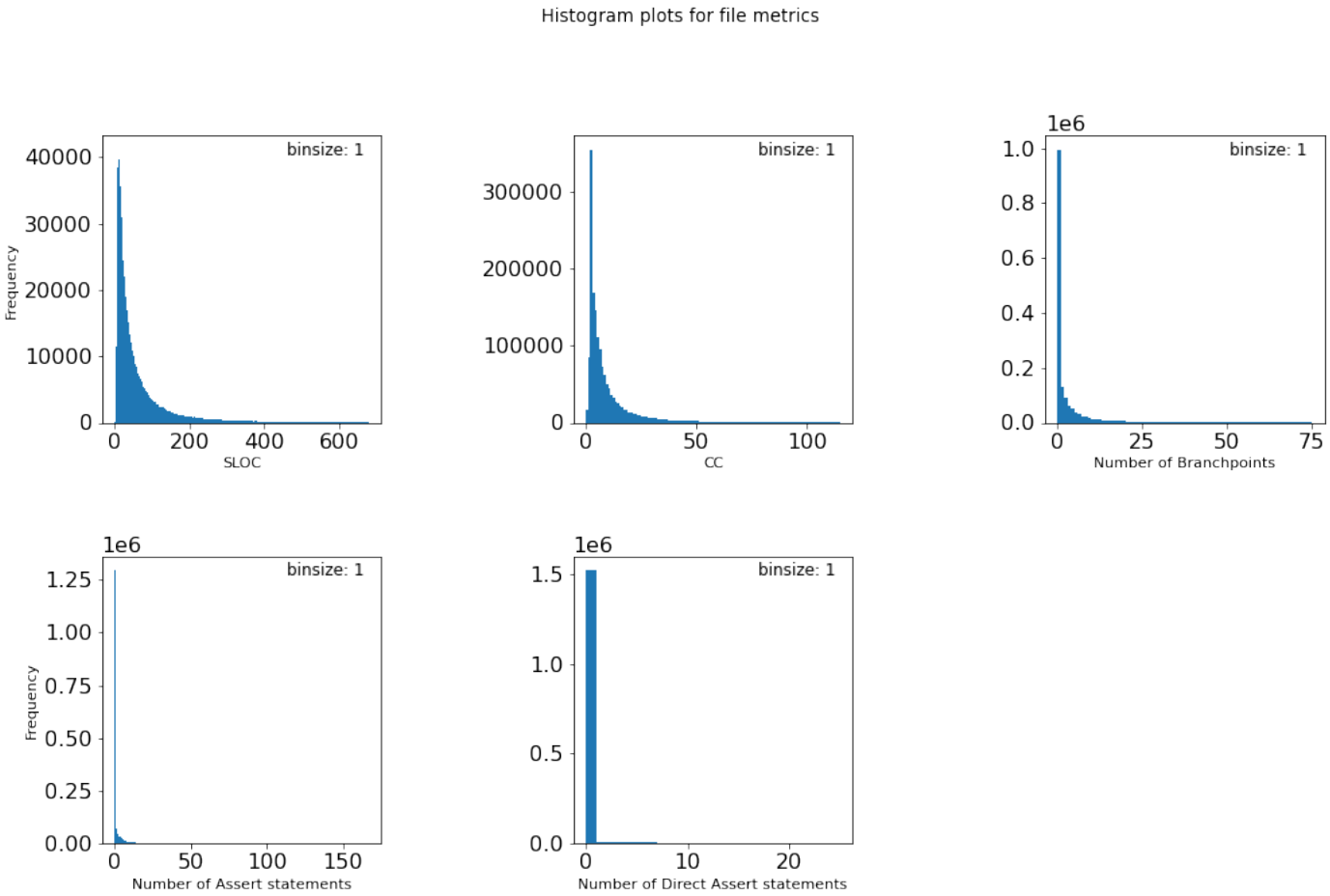

Figure 5.2: Histogram of file level metrics.

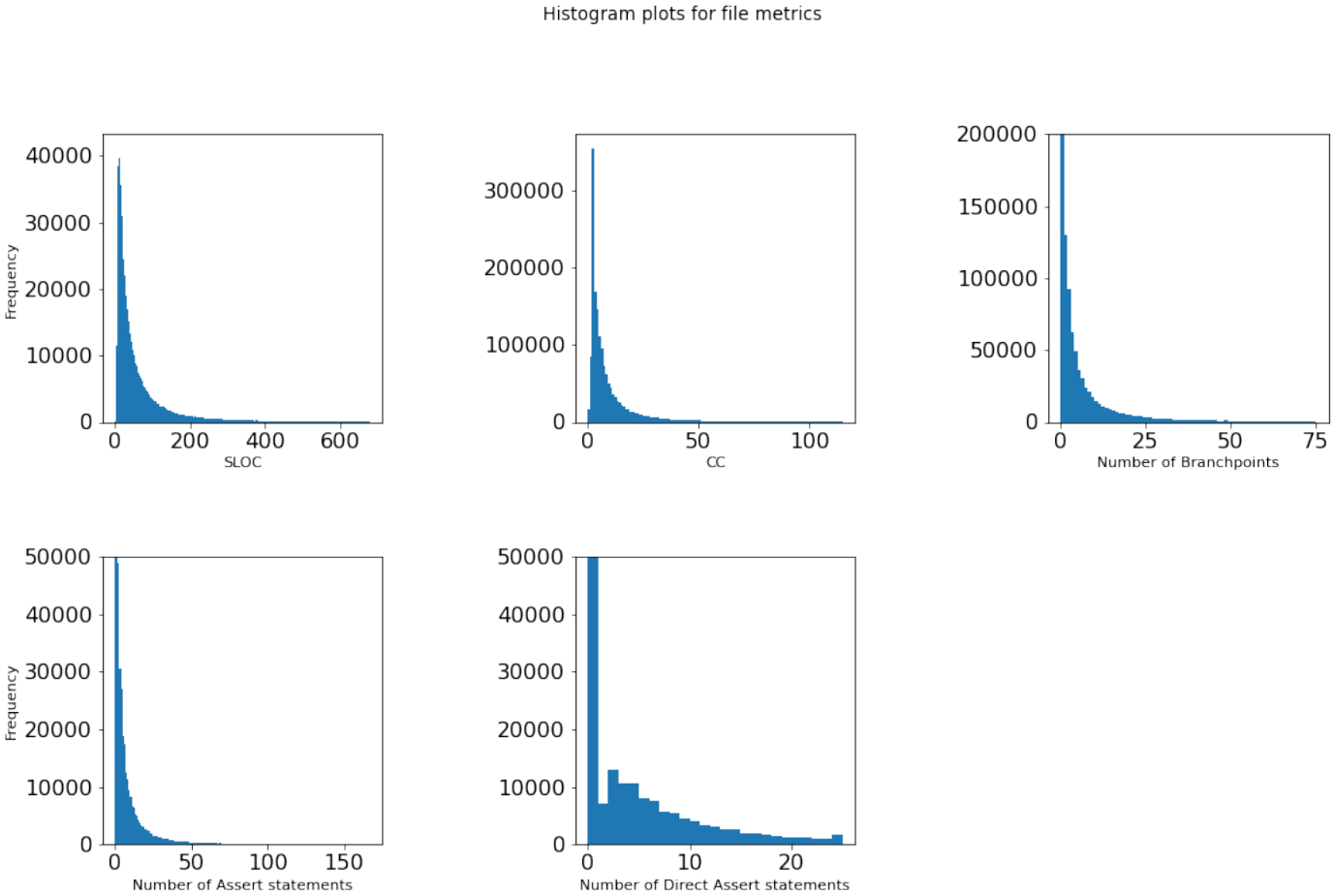

Figure 5.3: Histogram of file level metrics zoomed in.

5.4.3 System level

Tables

	min	5%	25%	50%	75%	95%	max	mean	skew	kurt
SLOC	34.0	835.05	7004.00	26323.0	86216.50	568331.15	3749092.0	125294.69	5.42	38.21
CC	6.0	104.35	997.00	4008.5	13074.50	87049.00	650534.0	19603.92	5.91	45.84
Branchpoints	0.0	23.80	278.25	1197.0	4628.00	31507.80	430953.0	8088.08	8.45	97.57
asserts	0.0	3.45	96.00	512.5	2223.75	19460.85	122049.0	3870.39	5.96	43.64
Direct asserts	0.0	0.00	3.25	123.5	825.00	6815.85	86309.0	1712.70	8.01	76.47

Table 5.5: Descriptive statistical metrics for Systems in the dataset.

SLOC	Frequency
431.0	4
32759.0	4
15591.0	4
14278.0	3
33236.0	3

CC	Frequency
63.0	4
66.0	3
43.0	3
5216.0	3
49.0	3

Branchpoints	Frequency
15.0	9
27.0	5
14.0	5
48.0	5
13.0	4

Asserts	Frequency
0.0	31
4.0	9
2.0	9
10.0	8
7.0	8

Direct asserts	Frequency
0.0	232
1.0	11
22.0	7
8.0	7
13.0	6

Table 5.6: Frequency tables for System level metrics.

Plots

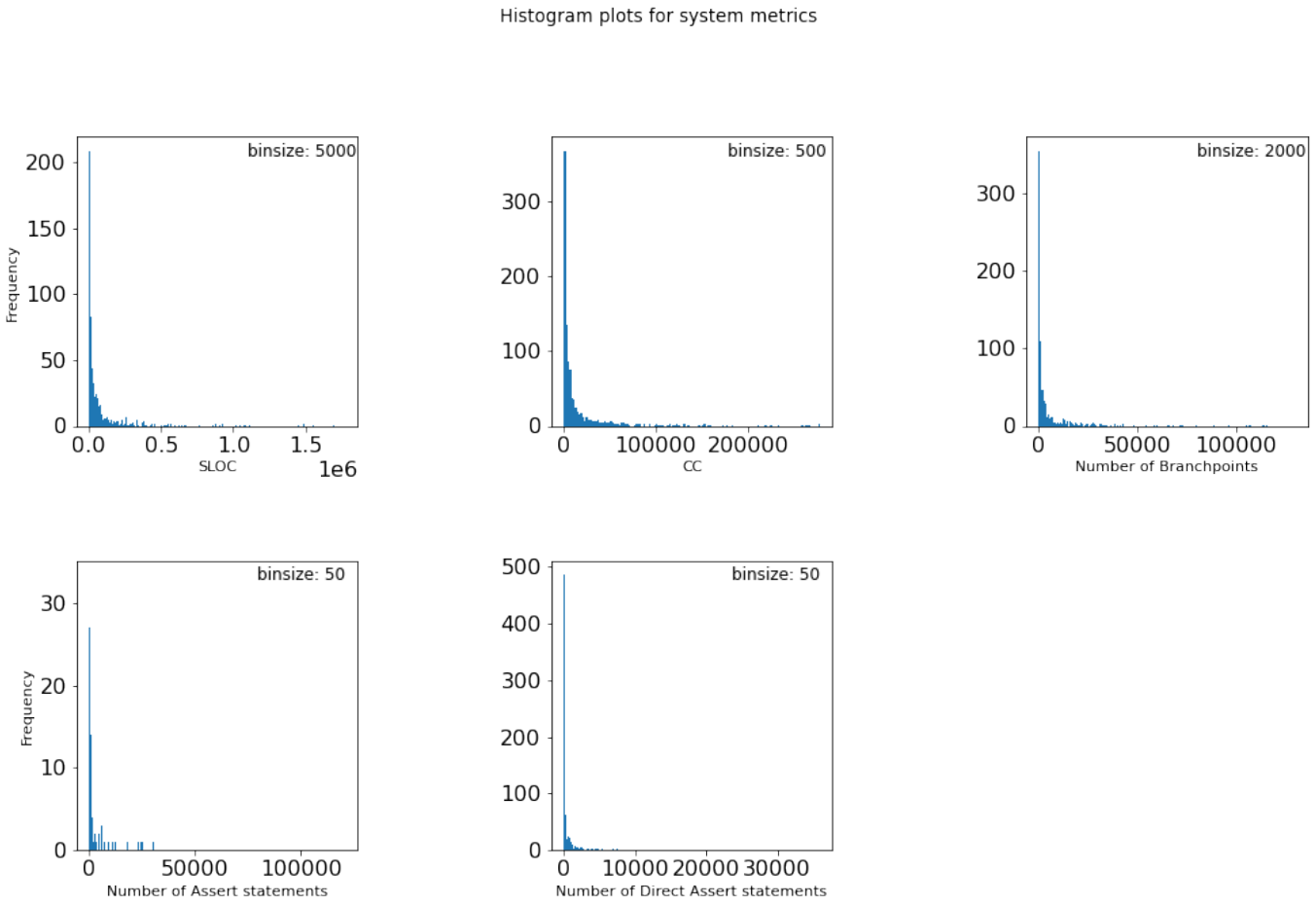

Figure 5.4: Histogram of system level metrics.

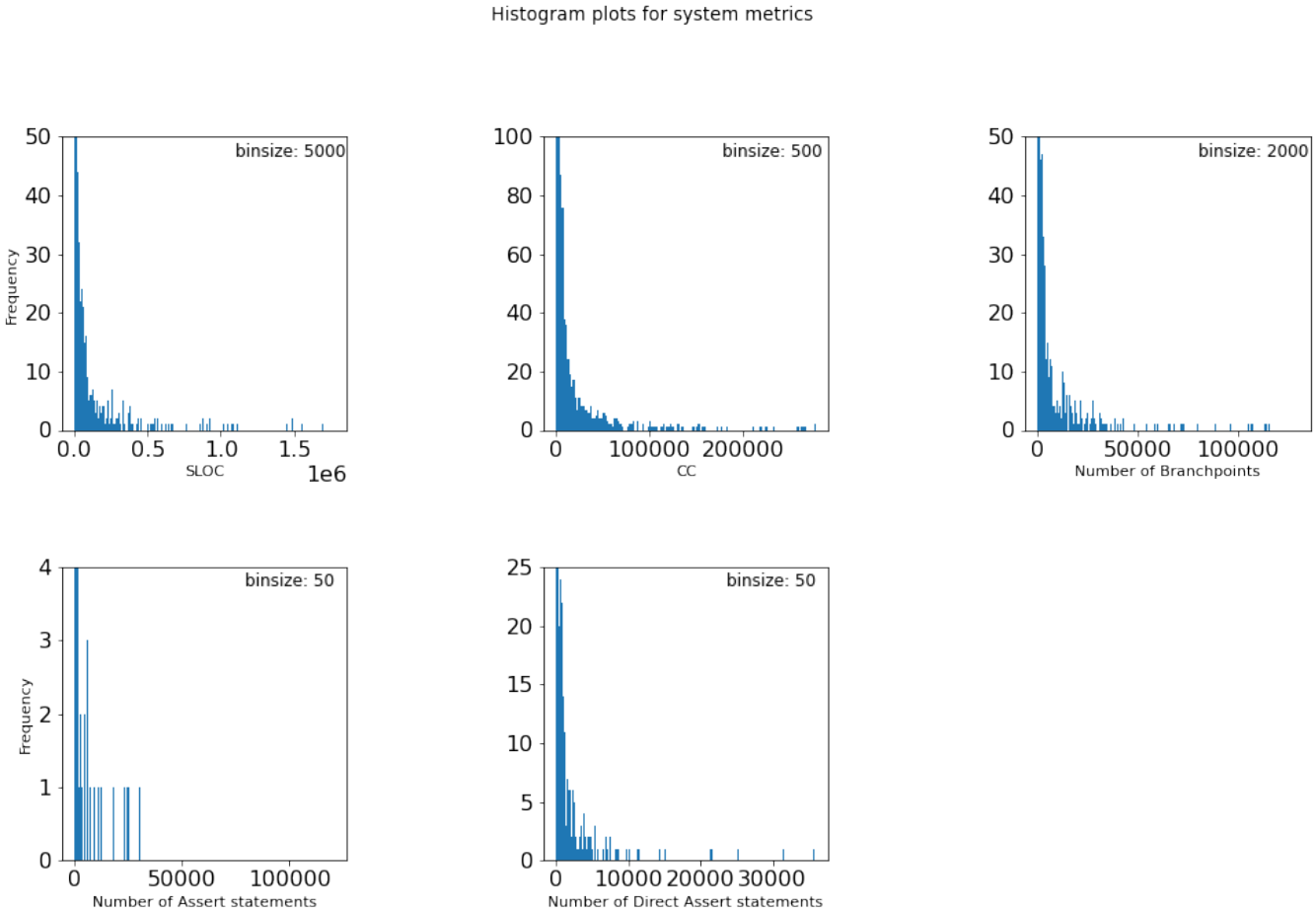

Figure 5.5: Histogram of system level metrics zoomed in.

5.5 Analysis

In this section we will analyze the tables, and we will demonstrate how values we think are interesting translate into code.

5.5.1 Interesting values

Unit level

At the unit level, we primarily consider the minimum for the SLOC metric to be interesting. This is an interesting value because these units are the smallest methods we encounter in our dataset, and the value is also 0. Because a method declaration is in itself a SLOC, we are very interested in what a 0 SLOC method looks like.

```
public interface UserDetails extends Serializable {

  boolean isAccountNonLocked();

  ...

}
```

Listing 5.1: An example of a method with 0 SLOC [20].

In code snippet 5.1 we see an example unit with 0 SLOC. The SAT does not count lines of code for interface methods, so all interface methods have a SLOC value of 0. According to the Oracle Java tutorials, interface methods are standard abstract methods [21]. For this reason, we did not filter these units from the dataset.
In the frequency table 5.1, we see that 3 SLOC is the mode at the unit level. Since this is the most common value, we want to know what it looks like in code. We take a look at how this looks in a code example.

```
protected UserDetailsService getUserDetailsService() {
  return this.userDetailsService;
}
```

Listing 5.2: An example of a method with 3 SLOC [22].

In code snippet 5.2 we see a getter that has 3 SLOC. The many getters and setters that Java and CSharp contain cause the unit-level mode for SLOC to be 3.
Furthermore, we also see in the frequency table 5.1 that methods with 1 SLOC are common. This is an interesting value, given that the method declaration is already a line in itself. We find out what a method with 1 SLOC looks like.

```
public class Api {
  public void doStuff() {}
}
```

Listing 5.3: An example of a method with 1 SLOC [23].

In code snippet 5.3 we see that it is indeed a method without a body. Only the method signature is counted in the number of SLOC for this unit.
For the remaining two metrics, we see no interesting values. The most common value is 1 CC, which translates to methods that do not contain branch points. In code snippet 5.2 we see an example of this.

File level

At the file level, we find the minimum SLOC value to be interesting, as it is 0. We would expect this to be an empty file.

```
/**
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

Listing 5.4: An example of a file with 0 SLOC [24].

As we see in code snippet 5.4, this is indeed a file with 0 SLOC, since the file only contains a multi-line comment.
We also see that files exist with 1 SLOC. This is remarkable because a class declaration by itself is 1 SLOC. We therefore examine whether this is a file with only a class declaration.

```
public class Other {}
```

Listing 5.5: An example of a file with 1 SLOC [25].

In code snippet 5.5 we see that our assumption is confirmed. All other values need no further explanation.

System level

At the system level, we want to take a look at what a system is according to the SAT. We want to demonstrate that the SAT also registers parts of a system as a system, for this programming language.

system	SLOC	CC	Branchpoints	asserts	Direct asserts
Tensorflow	15979.0	2182.0	1098.0	845.0	797.0
Facebookyoga	17217.0	1073.0	370.0	7094.0	68.0

Table 5.7: Tensorflow and Facebook-Yoga system level metrics.

In table 5.7 we have extracted two systems from the dataset that highlight this issue. If we go to the respective Github pages for Tensorflow [26] and Facebook-Yoga [27], we see that the languages measured in the table do not account for the majority of the code. This gives the impression that the systems are smaller than they really are, as only the CSharp and Java code is measured.

5.5.2 Distributions

We see skewed long tail distributions in the histograms at the unit, file and system levels. We see that the majority of the data points show small values and that there are large outliers.
The large Skewness values we read in the tables 5.1, 5.3 and 5.5 confirm that we are indeed looking at rightly skewed distributions. The high Kurtosis values are indicators on large outliers in the data. All values greater than 2 or less than -2 are considered a significant deviation from a normal distribution [28].
We can confirm the first hypothesis, we indeed see longtailed distributions at the unit and file level. We substantiate this through the graphs we see in Figures 5.1, 5.2, and we see this even better in the zoomed in file plot in figure 5.3. In addition, we support this through the Skewness and Kurtosis values we read in Tables 5.1 and 5.3.
Because of the way our data is distributed, and the large outliers it contains, the mean differs significantly

from the median and mode. Distributions with high Skewness and Kurtosis values cause the mean to become larger than the median. We can see this in the Tables 5.1, 5.3 and 5.5. In the frequency tables 5.2, 5.4 and 5.6 we see that the mean also deviates from the mode, which are mostly in the low values. Because the mean differs from the mode and median, we find that the mean is not a representative description of the dataset, and the value is too large. We confirm hypothesis 2.

5.6 Conclusion

We have described through Histograms and Tables the distribution of the dataset. We see skewed long tail distributions in the histograms at the unit, file and system levels. We see that the majority of the data points show small values and that there are large outliers.
In addition, we used frequency tables, along with the histograms and the statistically descriptive values, to identify interesting values where we did not have a good idea how this translates to code. We explained these with examples above.
We were able to accept hypothesis 1 based on viewing the histograms and the corresponding Kurtosis and Skewness values indicating a rightly skewed long tail distribution.
Also, we were able to confirm hypothesis 2 by showing that the mean deviates from the mode and median. Due to the large outliers, it is larger. Also, this is something characteristic of a distribution with high Skewness and Kurtosis values.

Chapter 6

Comparative analysis

6.1 Introduction

Before moving on to deriving threshold values for the Test Code Quality Metrics, we will research how to split the corpus. In the SIG's maintainability study, we found that CSharp and Java projects were analyzed together, with the assumption that these languages differ little from each other [8]. In the Landman study [19], the claim was made that production and test code differed and all test code was removed from the corpus for that study. We will try to challenge these two assumptions in the context of determining threshold values.
Combining code categories can introduce problems for the accuracy of the threshold values, if the individual threshold values differ from each other. When the threshold values of the two code categories separately are much different from each other, the combined threshold value will be somewhere in between. Depending on the number of data points, it will then be closer to one of the two languages. In case one of the two code categories has more data points, the combined threshold value will be mainly influenced by the dominant code category. In the case of a large difference between the individual threshold values, this will cause the combined threshold value for the code category with fewer data points to become less accurate.

We will focus on answering the following questions in this section:

- Can Java and CSharp projects be taken together to arrive at accurate threshold metrics?
- Can production and test code be taken together to arrive at accurate threshold metrics?

To answer the above questions, we will compare the distributions of the two code categories. We do this by using the Gini coefficient and the Lorenz curves, which allows us to compare how fairly both code categories are distributed. In addition, we will make cumulative percentile plots, which correspond to the plots made for determining risk profiles in the Alves method [9]. These are both visual tools that we will use to detect differences between the two code categories. We then calculate the maximum value for each 10th percentile for the metric to compare between the two code categories, so that in this way we can make a quantitative judgment about the difference in threshold values between the two code categories. We will determine the main differences at the 70th, 80th, and 90th percentiles, as these are common percentiles where many metrics show variability, as explained in the Alves method [9].
We chose to determine the maximum value for each 10th percentile instead of calculating the percentiles that we can read in the percentile plots. We did this because the percentile plots are linearly interpolated, which allows us to read values that cannot be found in our dataset. By calculating the maximum values below a percentile, we only get values that are actually data points.
An improvement point on the methodology described above is to determine the threshold percentiles based on the variability they show, as the methodology of Alves describes [9]. We chose to use fixed percentiles, the 70th, 80th and 90th percentiles, for all metrics due to time constraints.

Due to the size of all the results, we have moved them to Appendix A. Below we have presented only the tables we used to answer the above questions.

6.2 Hypotheses

Hypothesis 1. *We expect Java and CSharp code to yield to approximately the same threshold values.*
We base our hypothesis on the assumption done in the Maintainability paper [8] and the fact that a lot of the code constructs that exist in Java also exist in CSharp. They share the same defenition for a branch point for example.

Hypothesis 2. *We expect production and test code to yield different threshold values.*
We base our hypothesis on the assumption done in the Landman paper [19] where the claim is made that Test code has different characteristics from Production code.

6.3 Method

We started by splitting the data based on four code categories:

- Java production code
- Java test code
- CSharp production code
- CSharp test code

We stored these code categories in Python Pandas DataFrames for further analysis. Then we plotted the Lorenz Curves for the code categories we want to compare. To compare Java and CSharp, we compare Java production code with CSharp production code and Java test code with CSharp test code. To compare Production code and CSharp code, we compare Java production code with Java test code and CSharp production code with CSharp test code.
After plotting all Lorenz curves, for the same code categories, we plotted the percentile plots where the code categories we want to compare are reflected in a plot. We created these percentile plots using the method described in the Alves paper [9]. We have done this for each metric, at the unit, file and system level. For each data point, we calculated a weight based on the number of SLOC. We then sorted all the data points by the metric we wanted to plot and summed the weights cumulatively, giving the largest data point the cumulative weight of 1.
Finally, we calculated the metric for each 10th percentile. We did this by querying the maximum value in our dataset that falls below the cumulative percent value. We can then compare the percentile tables we created for each metric for the two code categories. For this we calculate the difference, and this ultimately allows us to conclude whether Java, CSharp, Production and Test code should be separated when determining threshold values.

6.4 Java - CSharp production code compared

In this part of the analysis, we look for counter examples that show that the distributions between Java and CSharp production code for certain metrics significantly, which results in different threshold values. We present for the relevant metrics the tables for which each 10th percentile is the maximum metric over the percentage of SLOC.

6.4.1 Production

Unit level

By studying the Gini values and the percentile plots, we see that the biggest difference between Java and CSharp units is in the SLOC metric.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production lines_of_code	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2238.0
CSharp production lines_of_code	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	28.57%	93.57%

Table 6.1: Percentile values of Java production and CSharp production for the lines_of_code metric on a Unit level.)

In table 6.1 we see that between the 50th and 90th percentiles there is a minimum difference of 47%, with the CSharp code being in units with at least 47% more SLOC. In absolute numbers, this is not a large difference, but it can result in numerous Units coming into different risk profiles when assessing systems at the unit level. This makes the calibration of the thresholds inaccurate, as we then get thresholds that are between the accurate Java and CSharp thresholds.

File level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	1149.0
Differences	~	~	~	~	~	~	~	~	~	38.6%

Table 6.2: Percentile values of Java production and CSharp production for the asserts metric on a File level.

In Table 6.2, we see large differences between the number of assert statements we find in files between Java and CSharp production code. Whereas at least 90% of Java production code is in files with no assert statements, at least 60% of CSharp production code sits within files with less than or equal to 1 assert statements. In Java, the 70th, 80th and 90th percentiles would not yield beneficial or usable risk profiles, since all these values are 0.

System level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	8.0	30.0	100.0	421.0	1264.0	25081.0
CSharp production asserts	301.0	686.0	1676.0	2455.0	3567.0	5479.0	6907.0	10041.0	11276.0	21348.0
Differences	~	~	83700.0%	81733.33%	44487.5%	18163.33%	6807.0%	2285.04%	792.09%	17.49%

Table 6.3: Percentile values of Java production and CSharp production for the asserts metric on a System level.

At the system level, this difference that we saw at the file level in Table 6.2 is further magnified. We can see from the differences in table 6.3 at almost all percentiles that there is a clear difference between Java and CSharp production code when it comes to the number of assert statements we can find in a system.

6.4.2 Test code

Unit level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test lines_of_code	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test lines_of_code	7.0	11.0	16.0	23.0	31.0	47.0	80.0	126.0	188.0	3605.0
Differences	75.0%	83.33%	100.0%	130.0%	158.33%	193.75%	300.0%	350.0%	300.0%	126.44%

Table 6.4: Percentile values of Java test and CSharp test for the lines_of_code metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test McCabe	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test McCabe	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	193.44%

Table 6.5: Percentile values of Java test and CSharp test for the McCabe metric on a Unit level.

In Tables 6.4 and 6.5, we see the differences between Java and CSharp test code for the SLOC and CC metrics. We see large differences in both metrics, particularly in the 70th, 80th, and 90th percentiles. Here we see clear differences that combined derivation of threshold values between Java and CSharp code would lead to inaccurate results for the SLOC and CC metrics in test code.

File level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test lines_of_code	46.0	71.0	99.0	133.0	177.0	239.0	330.0	484.0	855.0	18933.0
CSharp test lines_of_code	53.0	86.0	126.0	178.0	253.0	367.0	558.0	901.0	1854.0	68502.0
Differences	15.22%	21.13%	27.27%	33.83%	42.94%	53.56%	69.09%	86.16%	116.84%	261.81%

Table 6.6: Percentile values of Java test and CSharp test for the lines_of_code metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test nr_of_branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	656.0
CSharp test nr_of_branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	82.0	2828.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	256.52%	331.1%

Table 6.7: Percentile values of Java test and CSharp test for the nr_of_branchpoints metric on a Files level.

At a file level, we see that Java and CSharp test code differ mainly when it comes to the SLOC and Branchpoint metrics. In Table 6.6, we see that the difference at the 70th, 80th and 90th percentiles is at least 69%. In Table 6.7, we see an even larger difference at the 80th percentile, which is 111%.

System level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1736.0	4697.0	8632.0	14063.0	21127.0	26800.0	49274.0	66636.0	92921.0	96968.0
CSharp test asserts	1609.0	4538.0	8011.0	15738.0	20400.0	29171.0	41071.0	50276.0	56122.0	85742.0
Differences	7.89%	3.5%	7.75%	11.91%	3.56%	8.85%	19.97%	32.54%	65.57%	13.09%

Table 6.8: Percentile values of Java test and CSharp test for the asserts metric on a System level.

If we look at the percentile plots at the system level for Java and CSharp test code at the system level we see the biggest differences in the assert metrics. We find the metrics values by 10th percentile in Table 6.8. Here we see that the differences we see here are smaller than the differences we saw at the file and unit levels.

6.5 Production and test code compared

6.5.1 Java

Unit level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production McCabe	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
Java test McCabe	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	536.07%

Table 6.9: Percentile values of Java production and Java test for the McCabe metric on a Unit level.

In Table 6.9, we see the differences between production and test code for Java in the CC metric. We see that this is at least 400% for the 70th, 80th, and 90th percentiles, which will therefore ensure a large difference between the individual thresholds and the combined thresholds for Java production and test code.

File level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production nr_of_branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	104.0	1963.0
Java test nr_of_branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	656.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	352.17%	199.24%

Table 6.10: Percentile values of Java production and Java test for the nr_of_branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	1529.0
Differences	~	~	~	~	~	~	~	~	~	84.44%

Table 6.11: Percentile values of Java production and Java test for the asserts metric on a File level.

At the file level, we see large differences in the number of branch points and the number of assert statements. For the number of branchpoints, in Table 6.10, we see a minimum difference of 350% in the 70th, 80th, and 90th percentiles. For the Assert metric, we even see that at least 90% of the SLOC in Java production code is in files without assert statements. Also, we see that at least less than 20% of the SLOC is in files without assert statements for the test code.

System level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production nr_of_branchpoints	2414.0	6268.0	12021.0	18061.0	25915.0	35034.0	73326.0	97166.0	139875.0	187433.0
Java test nr_of_branchpoints	245.0	563.0	1133.0	1780.0	2465.0	4401.0	7256.0	10829.0	18823.0	34532.0
Differences	885.31%	1013.32%	960.99%	914.66%	951.32%	696.05%	910.56%	797.28%	643.11%	442.78%

Table 6.12: Percentile values of Java production and Java test for the nr_of_branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	8.0	30.0	100.0	421.0	1264.0	25081.0
Java test asserts	1736.0	4697.0	8632.0	14063.0	21127.0	26800.0	49274.0	66636.0	92921.0	96968.0
Differences	~	~	431500.0%	468666.67%	263987.5%	89233.33%	49174.0%	15728.03%	7251.34%	286.62%

Table 6.13: Percentile values of Java production and Java test for the asserts metric on a System level.

At the system level, the differences are evident in almost all percentiles. We see large differences for the Branchpoints and Asserts metrics between Java production and test code.

6.5.2 CSharp

Unit level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production lines_of_code	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
CSharp test lines_of_code	7.0	11.0	16.0	23.0	31.0	47.0	80.0	126.0	188.0	3605.0
Differences	133.33%	57.14%	60.0%	64.29%	72.22%	88.0%	135.29%	152.0%	108.89%	20.17%

Table 6.14: Percentile values of CSharp production and CSharp test for the lines_of_code metric on a Unit level.

At the unit level, we see that the number of SLOC for CSharp production and test code differs mainly at the 70th, 80th and 90th percentiles, with at least 108%.

File level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production McCabe	4.0	7.0	11.0	18.0	27.0	42.0	69.0	121.0	272.0	5822.0
CSharp test McCabe	2.0	2.0	3.0	5.0	7.0	12.0	21.0	42.0	123.0	2972.0
Differences	100.0%	250.0%	266.67%	260.0%	285.71%	250.0%	228.57%	188.1%	121.14%	95.9%

Table 6.15: Percentile values of CSharp production and CSharp test for the McCabe metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production nr_of_branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	74.0	171.0	4349.0
CSharp test nr_of_branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	82.0	2828.0
Differences	~	~	~	~	1100.0%	950.0%	550.0%	289.47%	108.54%	53.78%

Table 6.16: Percentile values of CSharp production and CSharp test for the nr_of_branchpoints metric on a File level.

At the file level, we see large differences in the CC and Branchpoint metrics. In Table 6.15 we see at least a 121% difference between the two code categories for the CC metric and in Table 6.16 we see a difference of at least 108% for the Branchpoint metric in the 70th, 80th and 90th threshold percentiles.

System level

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production nr_of_branchpoints	2800.0	7958.0	12701.0	20300.0	30834.0	40252.0	72772.0	102699.0	117813.0	243970.0
CSharp test nr_of_branchpoints	256.0	760.0	1486.0	2359.0	3843.0	4963.0	8125.0	9188.0	48751.0	186983.0
Differences	993.75%	947.11%	754.71%	760.53%	702.34%	711.04%	795.66%	1017.75%	141.66%	30.48%

Table 6.17: Percentile values of CSharp production and CSharp test for the nr_of_branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	301.0	686.0	1676.0	2455.0	3567.0	5479.0	6907.0	10041.0	11276.0	21348.0
CSharp test asserts	1609.0	4538.0	8011.0	15738.0	20400.0	29171.0	41071.0	50276.0	56122.0	85742.0
Differences	434.55%	561.52%	377.98%	541.06%	471.91%	432.41%	494.63%	400.71%	397.71%	301.64%

Table 6.18: Percentile values of CSharp production and CSharp test for the asserts metric on a System level.

Also, at the system level, we see large differences in the Branchpoints and Asserts metrics between the two code categories, which we can find in Tables 6.17 and 6.18. These large differences create inaccurate combined threshold values of CSharp production and test code in this corpus, for metrics that are calibrated at the system level.

6.6 Conclusion

In the above study, we examined if there is a difference between CSharp and Java code and Test and Production code. We found differences at the unit, file and system level for some of the metrics examined between Java and CSharp code. We also saw that differences are present for production and test code. These differences lead us to conclude that Java, CSharp, Production and Test code should all be kept separate from each other to produce threshold values that are accurate.
In the tables above, we have presented the biggest differences between Java and CSharp, and between Production and Test code in our corpus. Based on this, we can answer both questions posed in the introduction in the negative. We reject our first hypothesis, using the large differences we see, for example, in the assert metrics for production code in file and system level, and, for example, the CC metrics at file level for test code. We can substantiate Hypothesis 2 with the examples we have given above. In particular, the Assert metric differs at the file and system level for Java and CSharp between production and test code.

6.6.1 Threats to validity

There are a number of threat to validations for this research that we would like to name. First, the 70th, 80th, and 90th percentiles are not universal values that are appropriate for every metric. They are often used in the literature because they are also appropriate percentiles for many of the metrics. These percentiles have only been tested for production metrics, so the percentiles for test code could be

different. We see this reflected in Table 6.11, where we see that the Assert metric for Java production code still lacks variation at the 70th and 80th percentiles. Thus, these threshold percentiles are not appropriate for these file-level metrics. This is a threat to the correctness of the conclusions drawn above, because we have compared the different code types on percentiles with low variability, which might not represent the real difference that exist between these two code types and give the impression that they are similar.
In addition, we see that the percentile plots are linearly interpolated. This interpolation affects the accuracy of the percentile values read, since the plot indicates values that do not exist in the data set. Because of this interpolation, the percentile data also contain floats, while the metrics are integers. To calculate the threshold values, we took this into account by looking at the maximum value within the data set below the 70th, 80th and 90th percentiles of code. So these are always values that are also found in the data set. This is a threat to the validity of the metric values used to compare the code types, since many of the analyzed metrics are integers.
Also, this study asses quantitative data qualitative. We have tried to limit this as much as possible by presenting quantitative values in addition to the plots, the percentage difference between the threshold values. However, no acceptable deviation has been determined for these quantitative values, making it difficult to determine when two values deviate significantly from each other. This is a threat to the correctness of the conclusions drawn from this experiment, since there is no ground truth and the results are open for interpretation.
Finally, we have the influence of selection bias on this study. The results of a study on one corpus do not necessarily apply to other corpora. The final result can be strongly influenced by the coincidental choice of data points in the corpus. We will address this threat to validity in our validation chapter in Chapter 9. We will do this by repeating the study multiple times on a random subset of this corpus, and then looking at the differences in the results. The selection bias introduces a threat to the validity of the comparison done between the code types above, since these estimations might only be true for our corpus and might not be true to other corpora. We expect that especially percentiles with very high variability might yield other results in other corpora.

Chapter 7

Correlation analysis

7.1 Introduction

In the correlation analysis, we are going to take a closer look at the proposed metrics in the SIG test code quality model to see how these metrics correlate with each other, and with other basic metrics. A high linear correlation between two metrics means that the metrics provide us with the same information. Having two highly correlated metrics in a model is not beneficial, as has been explained in the previous chapters. A high correlation indicates that existence of an underlying factor that explains both metrics, or that one explains the other.
We also use this to test whether the proposed metrics give us new insights compared to common simple metrics. In case of high correlation, the trade-off can be made to use the simple metric instead of the new metric. These simple metrics are often metrics that have been researched more and are also more familiar to the end user of the model.
The prototype metrics we will analyze are:

1. Unverified Branchpoints
2. Unverified CC
3. Assert/Branch point ratio
4. Assert/CC ratio
5. Production/test code ratio

The simple basic metrics against which we will test the prototype metrics are:

1. SLOC (production / test)
2. CC
3. Branchpoints
4. Asserts
5. Direct asserts

This leads to the following questions for this section:

- How do the proposed metrics correlate with each other?
- How do the proposed metrics correlate with the simple metrics?
- Which of the proposed metrics do not give us new information?

7.2 Hypotheses

Hypothesis 1. *We expect assert/CC and assert/branch metrics to have a high correlation with each other.*
The antecedent in both ratios are the same. The difference between the two ratios is in the consistent. The difference between CC and number of branch points at the file level is equal to the number of units in the file. So we see a clear relationship between the two metrics. From this we base our expectation that the two metrics will show a high correlation with each other.

Hypothesis 2. *We expect unverified assert (branch) and unverified assert (CC) metrics to have a high correlation with each other.*
Here we have a similar situation. In both metrics, the number of direct asserts is subtracted from another metric. In the case of unverified assert (branch), the number of asserts is subtracted from the number of branch points. In the case of unverified assert (CC), the number of asserts is subtracted from the CC. Since we subtract the same metric from two metrics that we know have a relationship with each other, we expect a high correlation between these metrics.

Hypothesis 3. *We expect the assert/CC metric to have a high correlation with the direct assert and CC metrics.*
Because the metrics are built from these two metrics, we believe that these direct assert and CC metrics directly affect the Assert/CC ratio and thus exhibit a high correlation.

Hypothesis 4. *We expect the assert/branch metric to have a high correlation with the direct assert and branchpoint metrics.*
Just as hypothesis 3, this metric is calculated using the direct assert and branchpoint metrics. Therefore these two metrics have a direct influence on the Assert/branch metric. This makes us expect a high correlation between these metrics.

Hypothesis 5. *We expect the production/test code ratio to have a high correlation with production and test SLOC.*
Because the production/test code ratio is a calculated metric from two simple metrics, we expect these two simple metrics, the SLOC in production code and the SLOC in test code, to directly affect this ratio. Therefore, we expect to see high correlations in between these metrics.

7.3 Method

With Python we have read the data from CSV files into four DataFrames, Java files, Java systems, CSharp files and CSharp systems. In the next step we have filtered all files and systems with 0 direct asserts. We have made this choice based on the fact that approximately 90% of our files for Java and CSharp have 0 direct asserts. These files would influence the correlations, by making the difference between the unverified branch point metric and the branch point metric for 90% of the files 0. Besides that, we will later on filter these 0 values for the threshold analysis as well.
Then we have calculated the pairwise Pearson correlation of all the metrics. We have created two new dataframes from these Pearson correlations for each of the four code categories to only contain the correlations we are interested in. The first DataFrames consists of the correlations between the test code metrics and the simple metrics, the second consists of the Pearson correlation of the test code metrics between each other.
We have made correlation tables from these DataFrames, which we will present in the results below. We have observed and analyzed these tables and answered the questions mentioned in the introduction based on the calculated Pearson correlations.

7.4 Results: File metrics

7.4.1 Java

Table 7.1: Pearson correlation for Java files between test code metrics and simple metrics (Total)

	lines_of_code	McCabe	nr_of_branchpoints	asserts	asserts_in_direct_test
assert/branch	-0.099921	-0.114691	-0.127230	-0.005568	0.292228
assert/McCabe	-0.103089	-0.105874	-0.080644	-0.015495	0.264436
unverified_assert_branch	0.731290	0.831372	0.896116	0.020333	0.037145
unverified_assert_McCabe	0.822888	0.902895	0.893246	0.028536	0.049681

Table 7.2: Coefficient of determination for Java files between test code metrics and simple metrics (Total)

	lines_of_code	McCabe	nr_of_branchpoints	asserts	asserts_in_direct_test
assert/branch	0.009984	0.013154	0.016187	0.000031	0.085397
assert/McCabe	0.010627	0.011209	0.006503	0.000240	0.069927
unverified_assert_branch	0.534785	0.691179	0.803023	0.000413	0.001380
unverified_assert_McCabe	0.677145	0.815219	0.797888	0.000814	0.002468

Table 7.3: Pearson correlation for Java files between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe
assert/branch	1.000000	0.769121	-0.091251	-0.141759
assert/McCabe	0.769121	1.000000	-0.065481	-0.116251
unverified_assert_branch	-0.091251	-0.065481	1.000000	0.945392
unverified_assert_McCabe	-0.141759	-0.116251	0.945392	1.000000

Table 7.4: Coefficient of determination for Java files between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe
assert/branch	1.000000	0.591547	0.008327	0.020096
assert/McCabe	0.591547	1.000000	0.004288	0.013514
unverified_assert_branch	0.008327	0.004288	1.000000	0.893765
unverified_assert_McCabe	0.020096	0.013514	0.893765	1.000000

7.4.2 CSharp

Table 7.5: Pearson correlation for CSharp files between test code metrics and simple metrics. (total)

	lines_of_code	McCabe	nr_of_branchpoints	asserts	asserts_in_direct_test
assert/branch	-0.072674	-0.093187	-0.106298	-0.020112	0.400784
assert/McCabe	-0.067825	-0.084173	-0.074591	-0.019850	0.445390
unverified_assert_branch	0.853561	0.913961	0.946681	0.262041	0.041341
unverified_assert_McCabe	0.891766	0.952839	0.941340	0.276548	0.048787

Table 7.6: Coefficient of determination for CSharp files between test code metrics and simple metrics. (total)

	lines_of_code	McCabe	nr_of_branchpoints	asserts	asserts_in_direct_test
assert/branch	0.005282	0.008684	0.011299	0.000404	0.160628
assert/McCabe	0.004600	0.007085	0.005564	0.000394	0.198372
unverified_assert_branch	0.728566	0.835325	0.896205	0.068666	0.001709
unverified_assert_McCabe	0.795246	0.907902	0.886120	0.076479	0.002380

Table 7.7: Pearson correlation for CSharp files between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe
assert/branch	1.000000	0.796654	-0.085892	-0.113396
assert/McCabe	0.796654	1.000000	-0.070878	-0.098849
unverified_assert_branch	-0.085892	-0.070878	1.000000	0.971354
unverified_assert_McCabe	-0.113396	-0.098849	0.971354	1.000000

Table 7.8: Coefficient of determination for CSharp files between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe
assert/branch	1.000000	0.634658	0.007377	0.012859
assert/McCabe	0.634658	1.000000	0.005024	0.009771
unverified_assert_branch	0.007377	0.005024	1.000000	0.943528
unverified_assert_McCabe	0.012859	0.009771	0.943528	1.000000

7.5 Analysis: File metrics

We discuss Java and CSharp correlations together because we see little difference between the two correlation tables. If we examine table 7.3, 7.4, 7.7 and 7.8 we see high r^2 values for unverified branchpoints and unverified CC with basically all simple metrics. The r^2 values of the Assert/branch and the Assert/CC metrics fall lower for all simple metrics. We also see few differences in the r^2 values between Assert/Branch and Assert/CC and also between unverified branch points and unverified CC.
If we compare the r^2 value of the metrics among themselves we see that unverified branchpoints and unverified McCabe show very r^2 high values. We also see these high values between Assert/branch and Assert/McCabe. Between the ratio and absolute metrics, for example Assert/branch and unverified McCabe, we see a moderate r^2 value.
Based on the r^2 with simple metrics, we conclude that the absolute metrics, unverified branchpoints and unverified McCabe, provide little new information compared to the simple metrics, and see little reason to use these new metrics instead of the simple metrics. The ratio metrics have lower r^2 values and are therefore the better option.
Based on the r^2 values between the test code metrics themselves, we conclude that it is not a good idea to use both the absolute metrics, or both the ratio metrics. The metrics from the same category, absolute and ratio, have a high correlation with each other and thus provide us with the same information. Because of the medium r^2 values between the metrics from the different categories, we think it is wise to use only 1 of them.

7.6 Results: System metrics

7.6.1 Java

Table 7.9: Pearson correlation for Java systems between test code metrics and simple metrics (total)

	lines_of_code_prod	lines_of_code_test	McCabe_prod	nr_of_branchpoints_prod	asserts_prod	asserts_test	asserts_in_direct_test
assert/branch	-0.190467	-0.008387	-0.194996	-0.199455	-0.030822	0.015242	0.082863
assert/McCabe	-0.170792	0.040412	-0.176449	-0.181257	-0.002459	0.076874	0.156640
unverified_assert_branch	0.826080	0.253185	0.872762	0.942448	0.137529	0.228783	0.112775
unverified_assert_McCabe	0.966229	0.523403	0.984235	0.977336	0.280772	0.494087	0.382942
test/prod	-0.116606	0.163543	-0.127328	-0.147913	0.013813	0.167582	0.146696

Table 7.10: Coefficient of determination for Java systems between test code metrics and simple metrics (total)

	lines_of_code_prod	lines_of_code_test	McCabe_prod	nr_of_branchpoints_prod	asserts_prod	asserts_test	asserts_in_direct_test
assert/branch	0.036278	0.000070	0.038023	0.039782	0.000950	0.000232	0.006866
assert/McCabe	0.029170	0.001633	0.031134	0.032854	0.000006	0.005910	0.024536
unverified_assert_branch	0.682408	0.064102	0.761714	0.888207	0.018914	0.052341	0.012718
unverified_assert_McCabe	0.933599	0.273951	0.968719	0.955185	0.078833	0.244122	0.146644
test/prod	0.013597	0.026746	0.016212	0.021878	0.000191	0.028084	0.021520

Table 7.11: Pearson correlation for Java systems between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe	test/prod
assert/branch	1.000000	0.864013	-0.228424	-0.231233	0.630935
assert/McCabe	0.864013	1.000000	-0.235291	-0.226291	0.646401
unverified_assert_branch	-0.228424	-0.235291	1.000000	0.934446	-0.202656
unverified_assert_McCabe	-0.231233	-0.226291	0.934446	1.000000	-0.170422
test/prod	0.630935	0.646401	-0.202656	-0.170422	1.000000

Table 7.12: Coefficient of determination for Java systems between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe	test/prod
assert/branch	1.000000	0.746518	0.052178	0.053469	0.398079
assert/McCabe	0.746518	1.000000	0.055362	0.051207	0.417835
unverified_assert_branch	0.052178	0.055362	1.000000	0.873189	0.041070
unverified_assert_McCabe	0.053469	0.051207	0.873189	1.000000	0.029044
test/prod	0.398079	0.417835	0.041070	0.029044	1.000000

7.6.2 CSharp

Table 7.13: Pearson correlation for CSharp systems between test code metrics and simple metrics (total)

	lines_of_code_prod	lines_of_code_test	McCabe_prod	nr_of_branchpoints_prod	asserts_prod	asserts_test	asserts_in_direct_test
assert/branch	-0.123650	0.052962	-0.125140	-0.121006	-0.093000	0.149089	0.251770
assert/McCabe	-0.079280	0.063362	-0.081172	-0.077211	-0.050510	0.163805	0.253107
unverified_assert_branch	0.925003	0.603632	0.935817	0.986835	0.840166	0.376638	0.154144
unverified_assert_McCabe	0.991801	0.702697	0.995303	0.975467	0.911789	0.522846	0.300556
test/prod	-0.106872	0.207650	-0.105235	-0.113117	-0.079954	0.270271	0.241261

Table 7.14: Coefficient of determination for CSharp systems between test code metrics and simple metrics (total)

	lines_of_code_prod	lines_of_code_test	McCabe_prod	nr_of_branchpoints_prod	asserts_prod	asserts_test	asserts_in_direct_test
assert/branch	0.015289	0.002805	0.015660	0.014642	0.008649	0.022228	0.063388
assert/McCabe	0.006285	0.004015	0.006589	0.005961	0.002551	0.026832	0.064063
unverified_assert_branch	0.855631	0.364371	0.875753	0.973843	0.705878	0.141856	0.023760
unverified_assert_McCabe	0.983668	0.493784	0.990629	0.951535	0.831359	0.273368	0.090334
test/prod	0.011422	0.043118	0.011074	0.012795	0.006393	0.073047	0.058207

Table 7.15: Pearson correlation for CSharp systems between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe	test/prod
assert/branch	1.000000	0.899621	-0.141396	-0.152598	0.520117
assert/McCabe	0.899621	1.000000	-0.098886	-0.106204	0.413398
unverified_assert_branch	-0.141396	-0.098886	1.000000	0.955005	-0.138291
unverified_assert_McCabe	-0.152598	-0.106204	0.955005	1.000000	-0.132357
test/prod	0.520117	0.413398	-0.138291	-0.132357	1.000000

Table 7.16: Coefficient of determination for CSharp systems between test code metrics (total)

	assert/branch	assert/McCabe	unverified_assert_branch	unverified_assert_McCabe	test/prod
assert/branch	1.000000	0.809318	0.019993	0.023286	0.270521
assert/McCabe	0.809318	1.000000	0.009778	0.011279	0.170898
unverified_assert_branch	0.019993	0.009778	1.000000	0.912034	0.019124
unverified_assert_McCabe	0.023286	0.011279	0.912034	1.000000	0.017518
test/prod	0.270521	0.170898	0.019124	0.017518	1.000000

7.7 Analysis: System metrics

Also, at the system level, we do not see major differences between Java and CSharp values and will discuss them together. At a system level we see, in the tables 7.10 and 7.14 that the r^2 values between unverified branchpoints, unverified CC and the simple metrics are much higher than the r^2 values between assert/branch, assert/CC and the simple metrics.
Looking at the r^2 values in tables 7.12 and 7.16 we see that the values between Assert/branch and Assert/CC and also between unverified branchpoints and unverified CC is high. On this we base our conclusion that it is not a good idea to have the two ratio metrics and the two absolute metrics both in one model. We see that between these metrics categories moderate r^2 are present, which leads us to conclude that it is better to choose 1 metric for in the model.
Due to the correlations between unverified branchpoints, unverified CC and the simple metrics, we conclude that the unverified branchpoints and unverified CC are not suitable metrics for in the model. They add additional complexity, but provide little new information compared to these simple metrics.
Against our expectations, we see that the production/test code metric exhibits low r^2 values with the production and test SLOC. Between these metrics, we expected higher r^2 values. The production/test code metric shows very low values for all calculated correlations and r^2 values, leading us to conclude that the metric is not redundant.

7.8 Conclusion

As we explained above in our analysis, we calculated the correlations and r^2 values between the SIG test code quality metrics and the simple metrics to find out which metrics are not appropriate for in the model. Here we were able to find that the ratio metrics (Assert/branchpoint, Assert/CC) correlate less with the simple metrics than the absolute metrics (unverified branchpoints, unverified CC). We also found that the absolute metrics correlate a lot with each other, and the ratio metrics also correlate a lot with each other. Between the ratio and absolute metrics, we find a moderate correlation. This leads us to conclude that we want to include only one of the four metrics in the SIG test code quality model. Because the ratio metrics correlate less with the simple metrics than the absolute metrics, we conclude that the ratio metrics are better options for in the model relative to the absolute metrics. These conclusions apply to both File and System levels.
The test/production code ratio metric exhibits low correlations and r^2 values with all other metrics. Against our expectations, correlations with production SLOC and test SLOC are also low.

We accept hypothesis 1 based on the correlations between Assert/CC and Assert/branch for Java at the file level (0.769121), for CSharp at the file level (0.796654), for Java at the system level (0.864013), and for CSharp at the system level (0.899621).

We also accept hypothesis 2 based on the correlations between unverified branch points and unverified CC for Java at the file level (0.945392), for CSharp at the file level (0.971354), for Java at the system level (0.934446), and for CSharp at the system level (0.955005).
Next, we reject hypothesis 3 based on the correlations between the Assert/CC and the CC metrics and direct asserts based on the low correlations on file level: -0.105874 and 0.264436 for Java, -0.084173 and 0.445390 for CSharp, and also because of the low correlations on a system level: -0.176449 and 0.156640 for Java and -0.081172 and 0.253107 for CSharp.
The correlations between the Assert/branchpoint ratio and the branchpoint and direct asserts metrics is -0.127230 and 0.292228 for Java on a file level, -0.106298 and 0.400784 for CSharp on a file level, -0.199455 and 0.082863 for Java on a system level and -0.121006 and 0.251770 for CSharp on a system level. Based on these moderately high correlations on a file level and low correlations on a system level, we reject hypothesis 4.
Finally, we reject hypothesis 5 based on the low correlations that this metric has with all other metrics, both on a file and system level.

7.8.1 Threats to validity

Our biggest threat to validity is selection bias. We calculated these correlations across the entire dataset, but we do not know if these correlations are also found across other Java and CSharp corpora. These results may be specific and exclusive to this corpus. To counter this, in the validation of this research, chapter 9, we will repeat the experiment with a random sample of this corpus, in order to compare whether the results match and how much variability the iterations show. The selection bias introduces a threat to the validity of the calculated Pearson correlation between two variables, since these correlations might only be true for our corpus and might not be true to other corpora. We cannot say for certain that two variables that are highly correlated in our corpus, will be highly correlated for every corpus.
Another threat to validity is the choice to filter all files with 0 direct asserts from the dataset. We made this choice because a large portion of the files have no direct asserts. For the unverified branch and CC metrics, this ensures that in numerous cases these are equal to the branchpoint and CC metrics of those files, which of course would lead to a high correlation. This is a threat to the correctness of the calculated correlations, since the alteration of the data might have introduced correlations that would be different for the whole corpus. Also, the filtration of the corpus might have reduced the size to an extent that is too small to calculate correlations.

Chapter 8

Threshold analysis

8.1 Introduction

In this section, we will calculate the threshold values the new metrics: Assert/branch, Assert/CC, unverified branch, unverified CC and prod:test code ratio. We will focus on the Assert/McCabe ratio and the Test/production code metrics. These metrics are chosen based on our correlation study in Chapter 7. We will do this separately for Java and CSharp, based on our conclusions in Chapter 6.
At the file level, risk profiles are calculated for the Assert/branch metric using the Alves method [9]. At the system level, for the Assert/branch and the test/production code ratio, the system wide thresholds are calculated using the Baggen method [10].
We will also do this for the Assert/CC, unverified branch and unverified CC metrics, to identify the pain points in using the Alves and Baggen method. In this section, we will focus on the following questions:

- What is the risk profile of the Assert/branch metrics at the file level?
  - What are the threshold percentiles to calculate the risk profile?
- What are the system-wide threshold values of the Assert/branch and Test/production metrics to arrive at a 5-star score?
- What are the pain points in using the Alves and Baggen method for deriving threshold values?

8.2 Method

With Python we have read all the CSV files containing the SIG Data Warehouse data on a unit, file and system level into dataframes. We have calculated all the metrics on a file and system level and stored that as separate dataframe columns. Nest we calculate the risk profiles for the file level metrics. We do this according to the Alves method.
We first normalize the datapoints according to their SLOC. Next we sort the datapoints according to the metric we are calibrating. Finally, we plot the total cumulative metric using the cumulative weights. In the same plot we plot the cumulative metric of each system, to identify the variability.
Using the risk thresholds, we continue on determining the system level thresholds for each metric. We do this by calculating for each system the percentage of SLOC that falls within one of the risk values. We plot this data for each risk value into a graph, and calculate the 5th, 35th, 65th and 95th percentiles of the systems, sorted on the percentage of SLOC that falls within the risk value. This is done for all risk values, which we can combine into the threshold tables we will see below.

8.3 Risk profiles Assert/Branch & Assert/CC metrics

8.3.1 Assert/branch

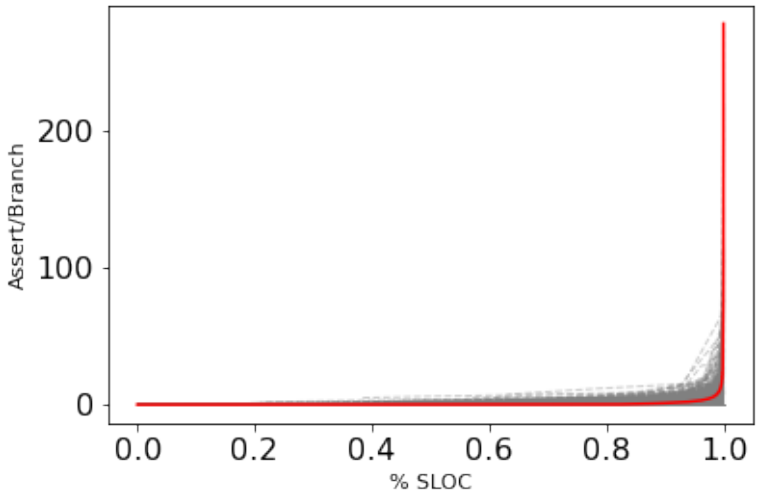

Figure 8.1: Risk profile plot for the Java Assert/Branch metric on a file level.

	Assert/Branch
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	0.0
70%	0.0
80%	0.0
90%	1.1

Table 8.1: Cumulative percentile values for Assert/Branch metric for Java files.

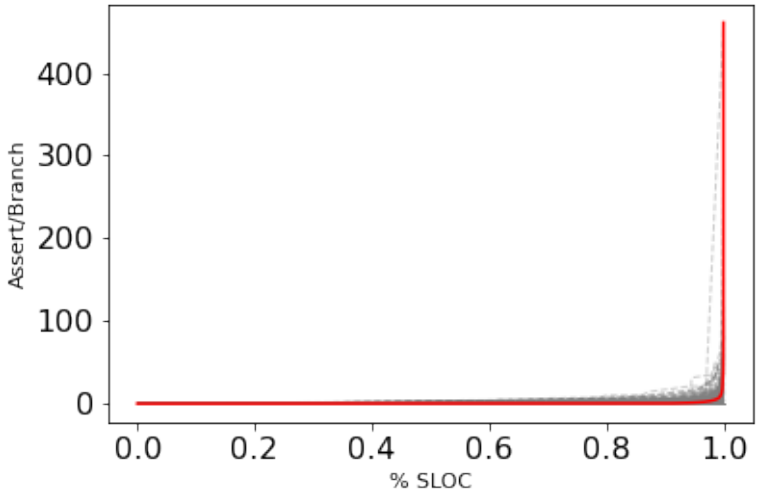

Figure 8.2: Risk profile plot for the CSharp Assert/Branch metric on a file level.

	Assert/Branch
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	0.0
70%	0.0
80%	0.0
90%	0.138

Table 8.2: Cumulative percentile values for Assert/Branch metric for CSharp files.

8.3.2 Assert/CC

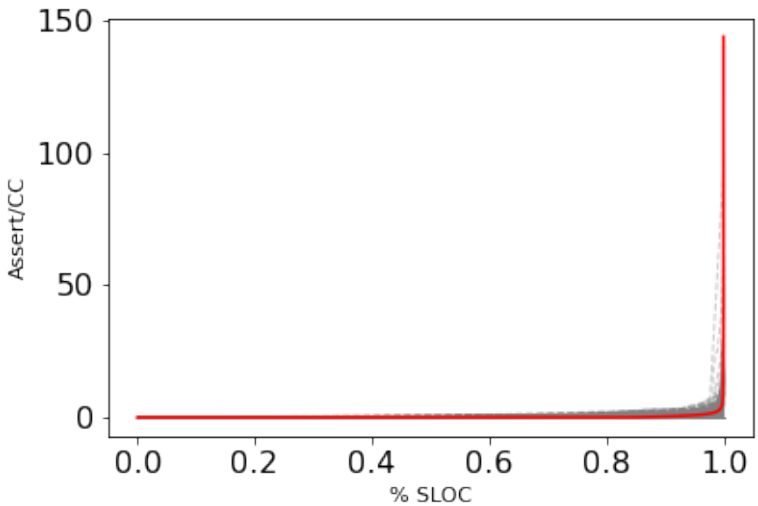

Figure 8.3: Risk profile plot for the Java Assert/CC metric on a file level.

	Assert/CC
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	0.0
70%	0.0
80%	0.0
90%	0.403

Table 8.3: Cumulative percentile values for Assert/CC metric for Java files.

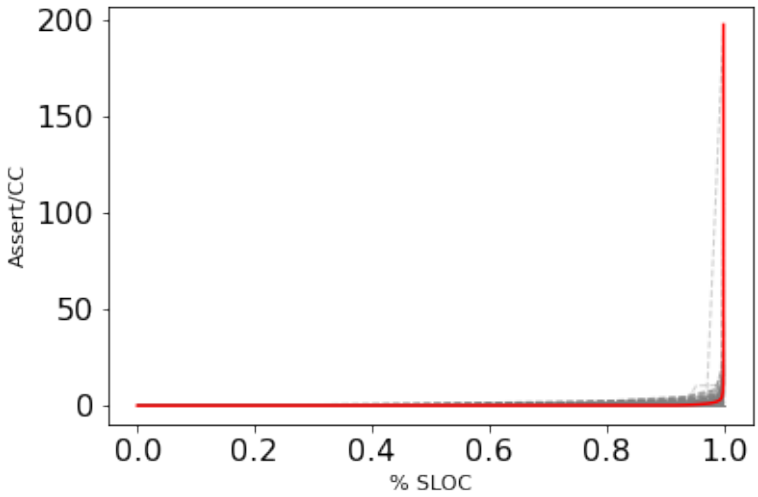

Figure 8.4: Risk profile plot for the CSharp Assert/CC metric on a file level.

	Assert/CC
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	0.0
70%	0.0
80%	0.0
90%	0.010

Table 8.4: Cumulative percentile values for Assert/CC metric for CSharp files.

8.3.3 Analysis

In the tables and percentile plots above, we immediately identify a pain point of the Alves method for determining risk profiles. Due to the large number of 0 values for the Direct assert metric, for Java and CSharp 80% of the SLOC are in files with an Assert/Branch and Assert/CC metric of 0. This is problematic, since we do not accept negative values for these metrics. This results in a non-useful risk profile. We can solve this by analyzing only files that have a Direct assert metric of greater than 0. This

does not affect the accuracy of the thresholds since the files without Direct asserts are automatically assigned a low score. With this method we base the risk profiles only on files that have actually been tested in the corresponding test files which comply with the naming conventions of Java and CSharp for test files.

8.4 Risk profiles Assert/Branch & Assert/CC metrics for filtered files

8.4.1 Assert/branch

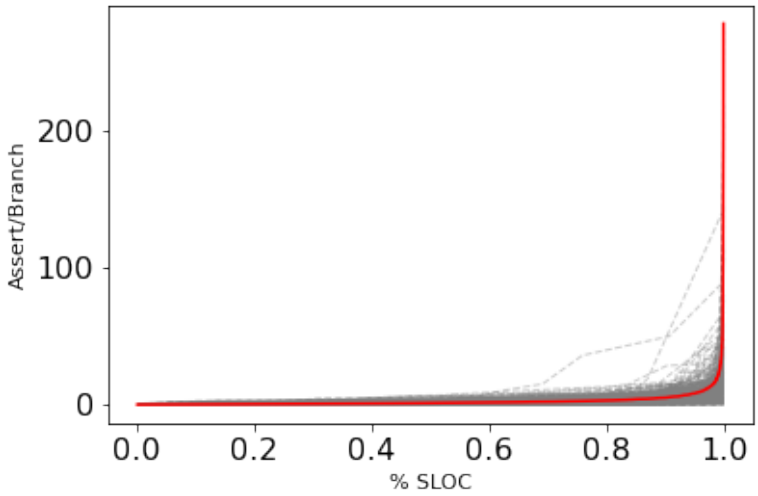

Figure 8.5: Risk profile plot for the Java Assert/Branch metric on a file level (filtered).

	Assert/Branch
10%	0.135
20%	0.314
30%	0.538
40%	0.805
50%	1.12
60%	1.527
70%	2.034
80%	3.0
90%	5.0

Table 8.5: Cumulative percentile values for Assert/Branch metric for Java files. (filtered)

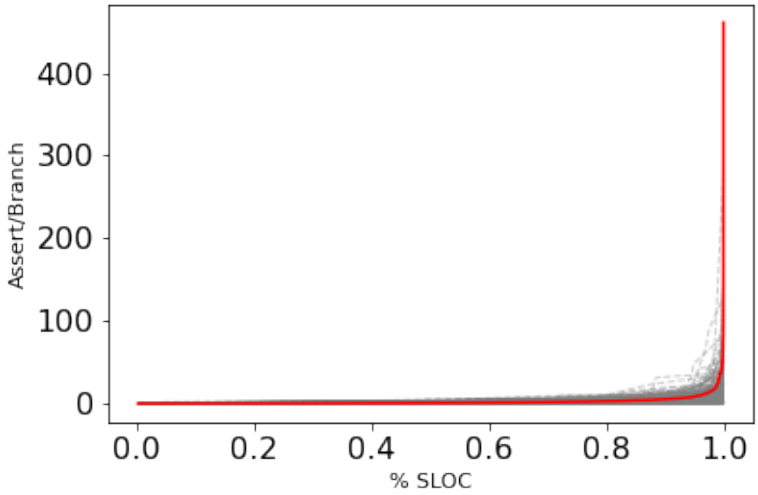

Figure 8.6: Risk profile plot for the CSharp Assert/Branch metric on a file level. (filtered)

	Assert/Branch
10%	0.052
20%	0.163
30%	0.308
40%	0.5
50%	0.830
60%	1.225
70%	1.909
80%	3.0
90%	5.077

Table 8.6: Cumulative percentile values for Assert/Branch metric for CSharp files. (filtered)

8.4.2 Assert/CC

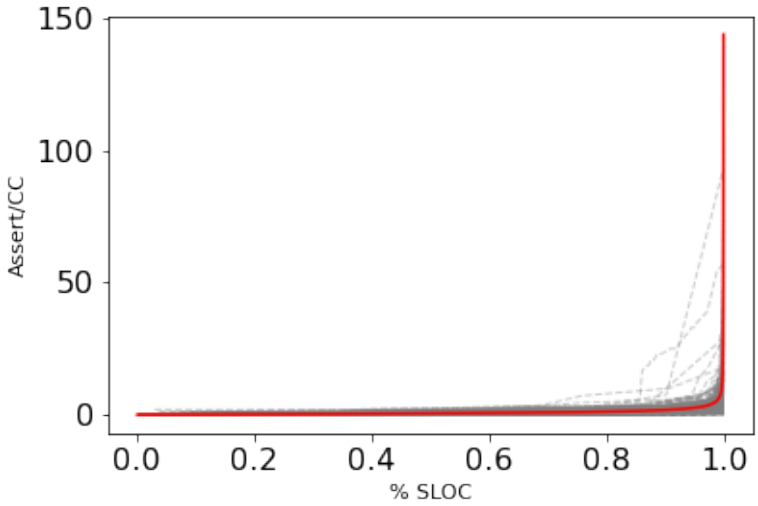

Figure 8.7: Risk profile plot for the Java Assert/CC metric on a file level. (filtered)

	Assert/CC
10%	0.077
20%	0.167
30%	0.268
40%	0.388
50%	0.512
60%	0.667
70%	0.889
80%	1.188
90%	1.808

Table 8.7: Cumulative percentile values for Assert/CC metric for Java files. (filtered)

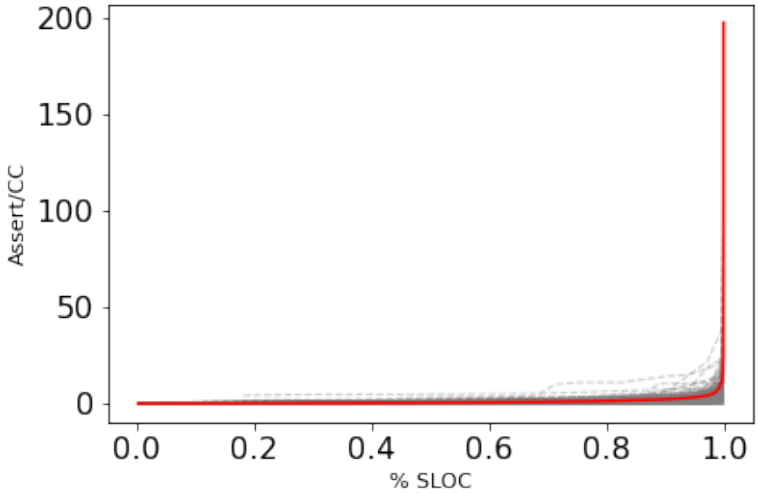

Figure 8.8: Risk profile plot for the CSharp Assert/CC metric on a file level. (filtered)

	Assert/CC
10%	0.039
20%	0.111
30%	0.2
40%	0.322
50%	0.476
60%	0.667
70%	0.962
80%	1.333
90%	2.222

Table 8.8: Cumulative percentile values for Assert/CC metric for CSharp files. (filtered)

8.4.3 Analysis

Above we see that our problem with the 0 values was solved by filtering out all files with 0 Direct asserts. We see in the plots that the variability is around the 70th, 80th and 90th threshold percentiles, which makes them suitable percentiles for determining the risk profile. In the tables we see very high values for the 70th, 80th and 90th percentiles. Here we have identified the second pain point. The Alves method is not suitable for creating risk profiles for metrics where the high values are desired. We determine the percentage of SLOC that is below a particular metric. Instead, we are looking for the percentage of SLOC that is above the metric value.
Here we can think of two solutions. First, we can mirror the plots so that the high values sit with the low percentiles. The problem with this solution is that the variability is mirrored along with it, putting it at the lower values as well. If we then pick the 10th, 20th and 30th percentiles for the metric values, we get the same values as before. If we stick to the 70th, 80th and 90th percentiles, they are not really based on anything anymore.
The other solution is to choose the 10th, 20th and 30th percentiles, without mirroring the plot. As explained above, we then have no methodology on which to base the percentile values, which makes this method unreliable. This leaves us with no suitable solution for this problem. We will derive the thresholds for these metrics on a system level with the Baggen method further below.

8.5 Risk profiles Unverified Branchpoints & Unverified CC

8.5.1 Unverified Branchpoints

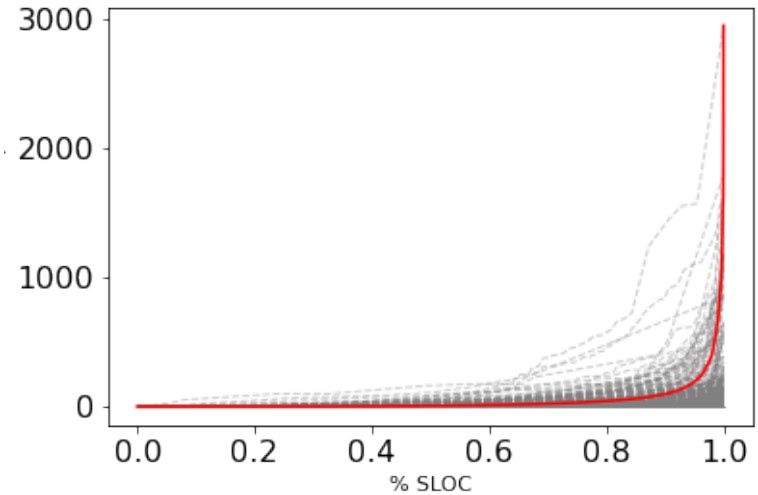

Figure 8.9: Risk profile plot for the Java Unverified Branchpoints metric on a file level.

	Unverified Branchpoints
10%	0.0
20%	0.0
30%	1.0
40%	3.0
50%	6.0
60%	12.0
70%	23.0
80%	43.0
90%	98.0

Table 8.9: Cumulative percentile values for the Unverified Branchpoints metric for Java files.

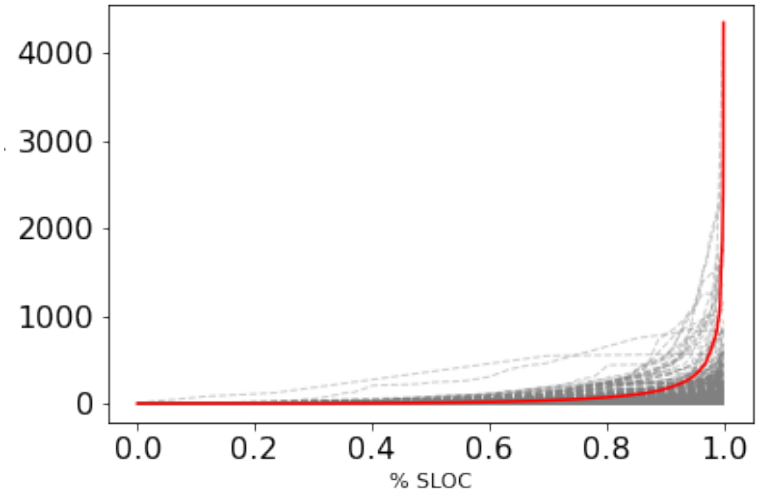

Figure 8.10: Risk profile plot for the CSharp Unverified Branchpoints metric on a file level.

	Unverified Branchpoints
10%	0.0
20%	0.0
30%	1.0
40%	4.0
50%	10.0
60%	19.0
70%	36.0
80%	71.0
90%	166.0

Table 8.10: Cumulative percentile values for the Unverified Branchpoints metric for CSharp files.

8.5.2 Unverified CC

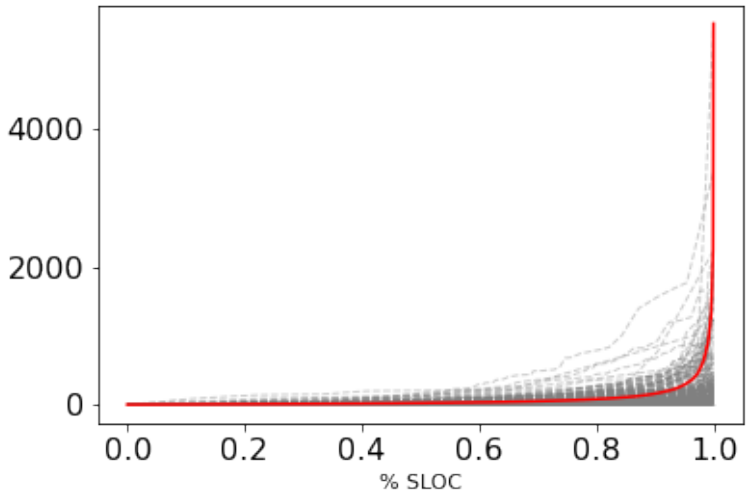

Figure 8.11: Risk profile plot for the Java Unverified CC metric on a file level.

	Unverified CC
10%	3.0
20%	6.0
30%	10.0
40%	15.0
50%	22.0
60%	33.0
70%	49.0
80%	79.0
90%	159.0

Table 8.11: Cumulative percentile values for the Unverified CC metric for Java files.

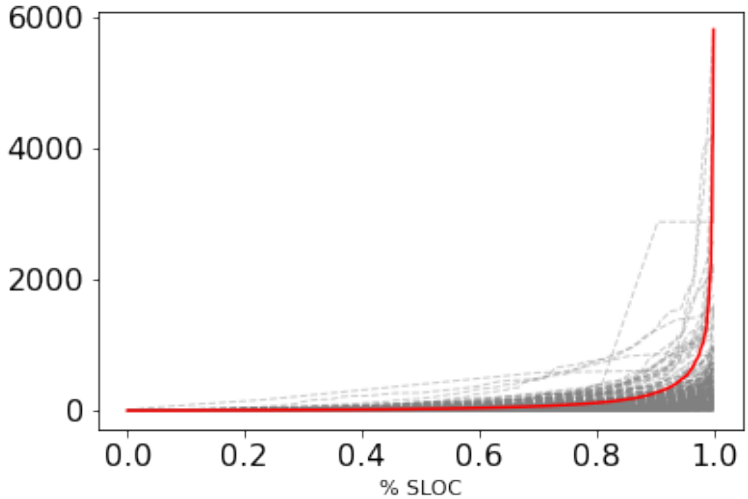

Figure 8.12: Risk profile plot for the CSharp Unverified CC metric on a file level.

	Unverified CC
10%	3.0
20%	6.0
30%	10.0
40%	16.0
50%	25.0
60%	40.0
70%	66.0
80%	117.0
90%	269.0

Table 8.12: Cumulative percentile values for the Unverified CC metric for CSharp files.

8.5.3 Analysis

For the unverified branchpoint and unverified CC metrics, we see that both problems we saw above have been solved. The 0 values of the Direct assert metric do not cause all of our metrics to become 0 for the percentiles, and the low values of the metrics are desirable, thus making the Alves method suitable for these metrics. So the problem with this metric is not the creation of a suitable risk profile, but the high correlations that we saw in Chapter 7.
The fact that we include all untested files in determining risk profiles and thresholds is also a problem. We know that the unverified branchpoints and unverified CC of untested files is always equal to the branchpoints and CC of the file. Therefore, we will also create risk profiles where all untested files are filtered out of the dataset. In this way we ensure that the model, which is intended to analyze test code, is only calibrated with files for which test code is also written.

8.6 Risk profiles Unverified Branchpoints & Unverified CC Filtered

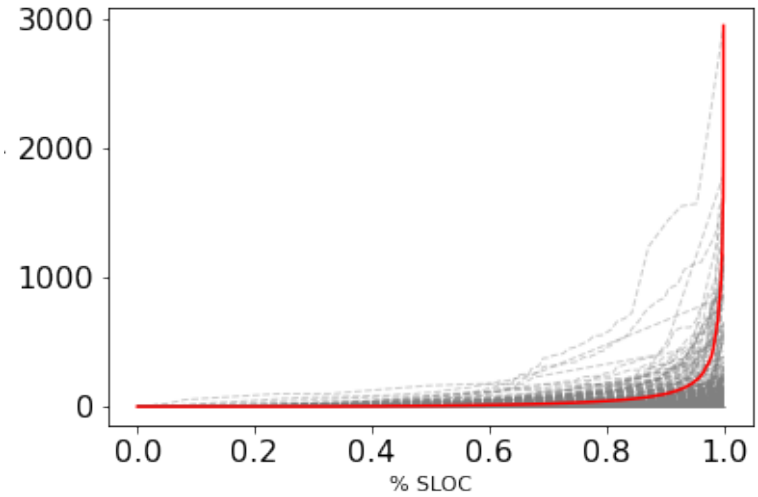

Figure 8.13: Risk profile plot for the Java Unverified Branchpoints metric on a file level. (filtered)

	Unverified Branch
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	1.0
70%	6.0
80%	19.0
90%	52.0

Table 8.13: Cumulative percentile values for the Unverified Branchpoints metric for Java files. (filtered)

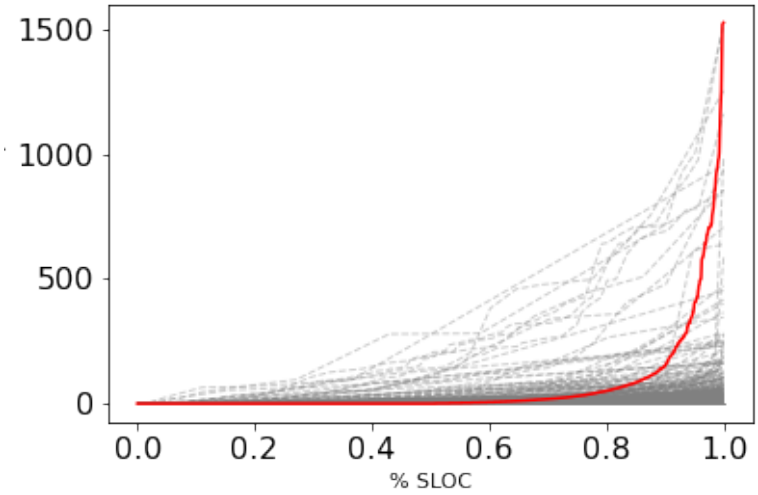

Figure 8.14: Risk profile plot for the CSharp Unverified Branchpoints metric on a file level. (filtered)

	Unverified Branch
10%	0.0
20%	0.0
30%	0.0
40%	0.0
50%	0.0
60%	6.0
70%	18.0
80%	51.0
90%	150.0

Table 8.14: Cumulative percentile values for the Unverified Branch metric for CSharp files. (filtered)

8.6.1 Unverified CC

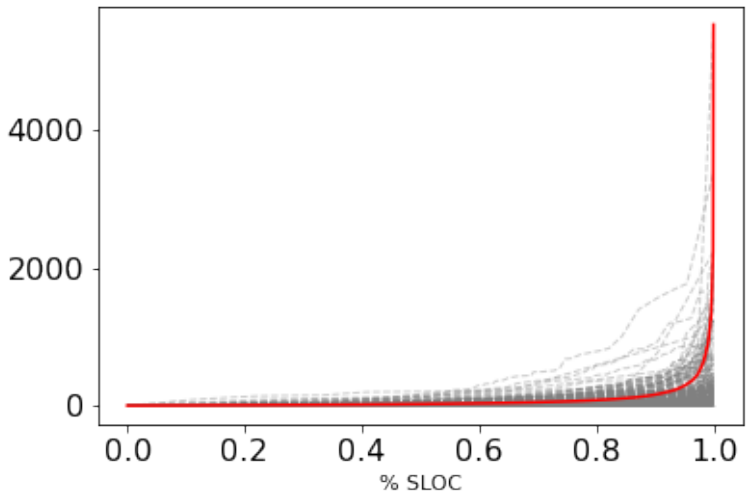

Figure 8.15: Risk profile plot for the Java Unverified CC metric on a file level. (filtered)

	Unverified CC
10%	0.0
20%	0.0
30%	2.0
40%	5.0
50%	11.0
60%	18.0
70%	30.0
80%	52.0
90%	107.0

Table 8.15: Cumulative percentile values for the Unverified CC metric for Java files. (filtered)

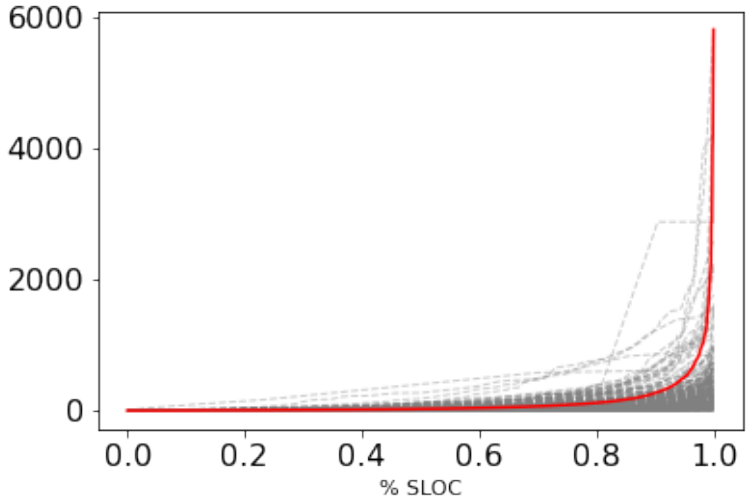

Figure 8.16: Risk profile plot for the CSharp Unverified CC metric on a file level. (filtered)

	Unverified CC
10%	0.0
20%	0.0
30%	1.0
40%	4.0
50%	10.0
60%	21.0
70%	43.0
80%	90.0
90%	227.0

Table 8.16: Cumulative percentile values for the Unverified CC metric for CSharp files. (filtered)

8.6.2 Analysis

So by filtering out all the untested files we see that at all percentiles have smaller unverified CC and unverified Branch points values. We feel this is more accurate since with the SIG Test code quality model was created to evaluate test code. Therefore, we calibrate the risk profiles only on test code that has actually been tested. This automatically ensures that untested code falls into higher risk profiles.

8.7 Thresholds for Assert/Branch & Assert/CC metrics

8.7.1 Assert/branch

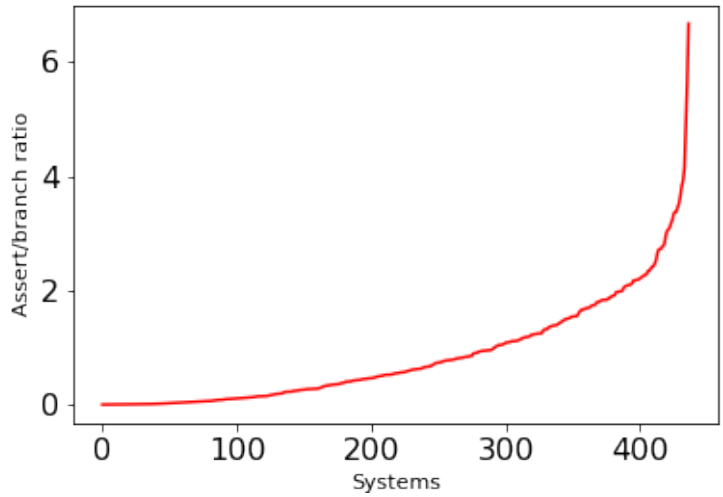

Figure 8.17: Threshold plot for the Java Assert/branchpoint metric on a file level.

	Assert/Branch
⋆⋆⋆⋆⋆	>2.719
⋆⋆⋆⋆	>0.939
⋆⋆⋆	>0.264
⋆⋆	>0.004
⋆	<

Table 8.17: Threshold values for the Assert/Branch metric for Java systems.

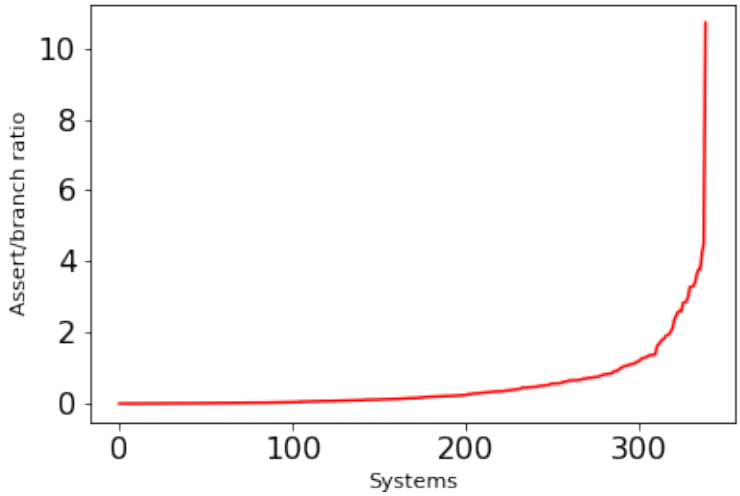

Figure 8.18: Threshold plot for the CSharp Assert/branchpoint metric on a file level.

	Assert/Branch
⋆⋆⋆⋆⋆	>2.471
⋆⋆⋆⋆	>0.339
⋆⋆⋆	>0.071
⋆⋆	>0.003
⋆	<

Table 8.18: Threshold values for the Assert/Branch metric for CSharp systems.

8.7.2 Assert/CC

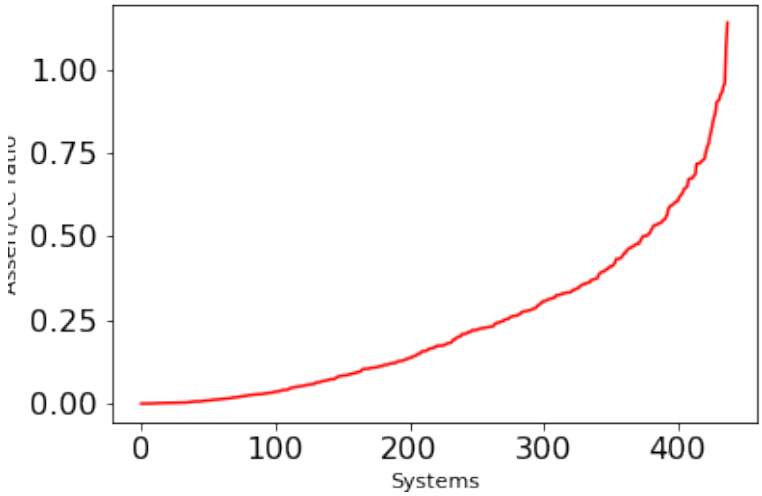

Figure 8.19: Threshold plot for the Java Assert/CC metric on a file level.

	Assert/CC
⋆⋆⋆⋆⋆	>0.719
⋆⋆⋆⋆	>0.270
⋆⋆⋆	>0.085
⋆⋆	>0.002
⋆	<

Table 8.19: Threshold values for the Assert/CC metric for Java systems.

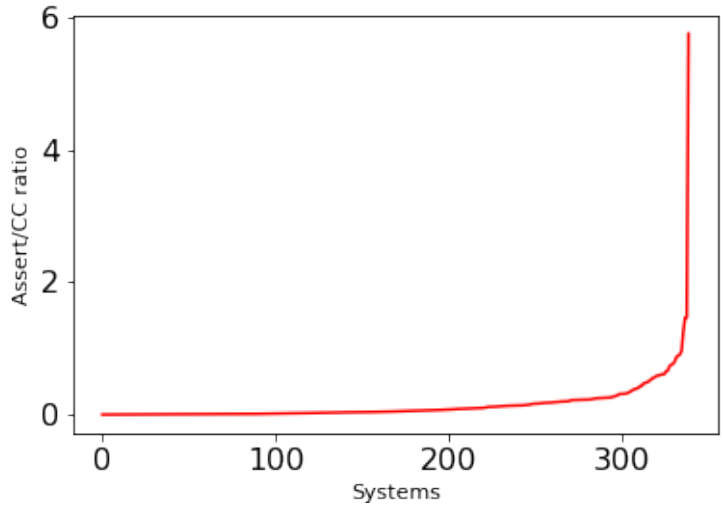

Figure 8.20: Threshold plot for the CSharp Assert/CC metric on a file level.

	Assert/CC
⋆⋆⋆⋆⋆	>0.590
⋆⋆⋆⋆	>0.100
⋆⋆⋆	>0.022
⋆⋆	>0.001
⋆	<

Table 8.20: Threshold values for the Assert/CC metric for CSharp systems.

8.7.3 Analysis

As we see above, we deviated from the standard Alves and Baggen method of determining thresholds to find that these metrics pose problems in risk profiling. We sorted the systems based on the metrics, and took the 5th, 35th, 65th, and 95th percentiles of systems as the threshold value. This way is independent of the variability of the metrics. This method is more commonly used for system level metrics for which risk profiles cannot be created and is similar to the method SIG used to calibrate the Duplication metric in the maintainability model [8]. The same method will be used to calibrate the thresholds for the production:test code ratio metric, since that is a system level metric.

8.8 Thresholds for unverified branchpoints and unverified CC.

8.8.1 Unverified branchpoints

	Moderate	High	Very High
⋆⋆⋆⋆⋆	76.75%	59.27%	45.09%
⋆⋆⋆⋆	99.77%	92.63%	83.23%
⋆⋆⋆	100%	100%	97.49%
⋆⋆	100%	100%	100%
⋆	-	-	-

Table 8.21: Threshold values for the unverified branchpoints metric for Java systems.

	Moderate	High	Very High
⋆⋆⋆⋆⋆	68.10%	52.40%	37.65%
⋆⋆⋆⋆	96.16%	85.77%	74.05%
⋆⋆⋆	100%	97.81%	91.09%
⋆⋆	100%	100%	100%
⋆	-	-	-

Table 8.22: Threshold values for the unverified branchpoints metric for CSharp systems.

8.8.2 unverified CC

	Moderate	High	Very High
⋆⋆⋆⋆⋆	64.00%	43.83%	25.57%
⋆⋆⋆⋆	95.82%	79.16%	61.37%
⋆⋆⋆	100%	95.90%	82.62%
⋆⋆	100%	100%	100%
⋆	-	-	-

Table 8.23: Threshold values for the unverified CC metric for Java systems.

	Moderate	High	Very High
⋆⋆⋆⋆⋆	59.30%	38.61%	22.62%
⋆⋆⋆⋆	90.73%	73.92%	56.34%
⋆⋆⋆	100%	91.32%	75.42%
⋆⋆	100%	100%	100%
⋆	-	-	-

Table 8.24: Threshold values for the unverified CC metric for CSharp systems.

8.8.3 Analysis

As we see again above, the unverified branch points and unverified CC are suitable for the Alves and Baggen method to arrive at threshold values. The tables cumulatively indicate what portion of the SLOC may fall into what risk profile to achieve that number of stars as a score. These are the threshold values for the unverified branch points and unverified CC metrics including all untested files. We motivated above why we think it is more accurate to calibrate thresholds only with tested files. We will do that below.

8.9 Thresholds for unverified branchpoints and unverified CC. (filtered)

8.9.1 Unverified Branchpoints

	Moderate	High	Very High
⋆⋆⋆⋆⋆	62.77%	41.55%	21.76%
⋆⋆⋆⋆	94.89%	79.11%	58.16%
⋆⋆⋆	100%	95.18%	78.57%
⋆⋆	100%	100%	100%
⋆	-	-	-

Table 8.25: Threshold values for the unverified Branchpoints metric for Java systems. (filtered)

	Moderate	High	Very High
⋆⋆⋆⋆⋆	56.21%	34.76%	18.43%
⋆⋆⋆⋆	89.06%	70.07%	49.36%
⋆⋆⋆	100%	88.87%	66.11%
⋆⋆	100%	100%	98.29%
⋆	-	-	-

Table 8.26: Threshold values for the unverified Branchpoints metric for CSharp systems. (filtered)

8.9.2 Unverified CC

	Moderate	High	Very High
⋆⋆⋆⋆⋆	49.94%	20.90%	3.82%
⋆⋆⋆⋆	84.24%	54.92%	18.77%
⋆⋆⋆	98.43%	76.34%	36.82%
⋆⋆	100%	100%	75.74%
⋆	-	-	-

Table 8.27: Threshold values for the unverified CC metric for Java systems. (filtered)

	Moderate	High	Very High
⋆⋆⋆⋆⋆	44.35%	19.57%	3.84%
⋆⋆⋆⋆	79.03%	50.29%	17.20%
⋆⋆⋆	94.09%	70.26%	31.72%
⋆⋆	100%	100%	66.80%
⋆	-	-	-

Table 8.28: Threshold values for the unverified CC metric for CSharp systems. (filtered)

8.9.3 Analysis

Above, we see that filtering out the untested files makes higher scores more difficult to obtain, by reducing the maximum percentages for each bin. We think that filtering the untested files makes the model more accurate, since the model is created to evaluate test code.

8.10 Thresholds for Test:production code ratio metric

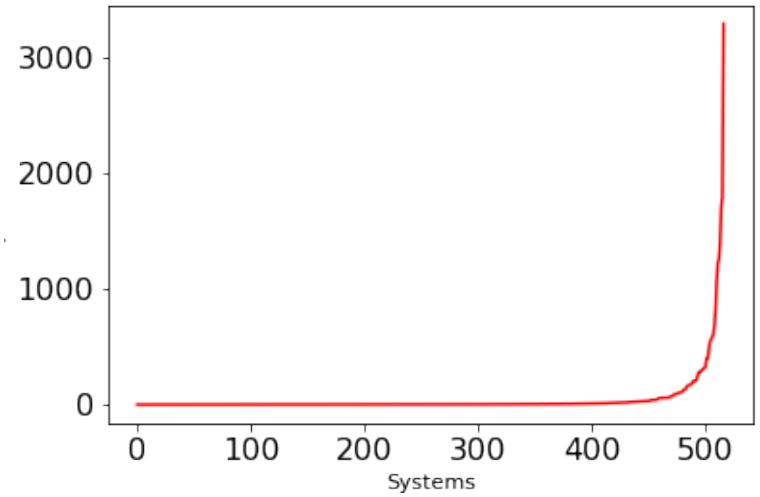

Figure 8.21: Threshold plot for the Java Test:production code ratio metric.

	Test:Production code ratio
⋆⋆⋆⋆⋆	>201
⋆⋆⋆⋆	>3.89
⋆⋆⋆	>1.44
⋆⋆	>0.64
⋆	<0.64

Table 8.29: Threshold values for the Test:production code ratio metric for Java systems.

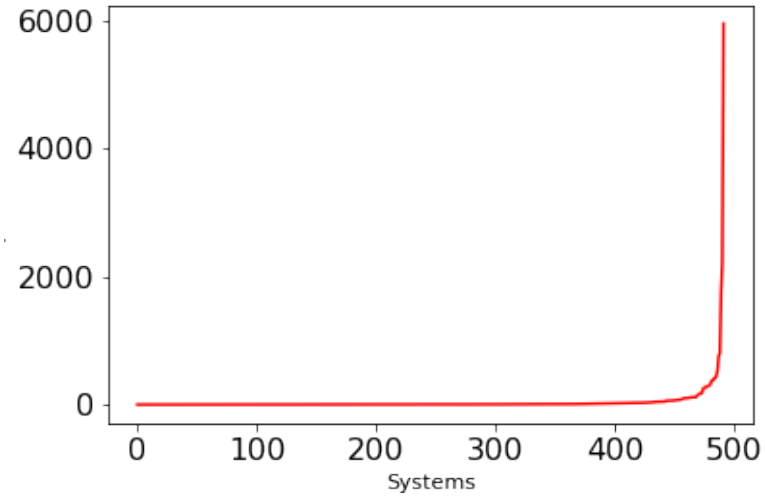

Figure 8.22: Threshold plot for the CSharp Test:production code ratio metric.

	Test:Production code ratio
⋆⋆⋆⋆⋆	>115.58
⋆⋆⋆⋆	>4.92
⋆⋆⋆	>1.78
⋆⋆	>0.52
⋆	<0.52

Table 8.30: Threshold values for the Test:production code ratio metric for CSharp systems.

8.10.1 Analysis

Because the Test:production code ratio metric is a system level metric, no risk profiles have been created for it. We departed from the standard Alves and Baggen method for establishing threshold values and adapted the Baggen method. We sorted all systems based on the Test:production code metric, from which the plots were created in Figures 8.21 and 8.22 are created. From this plots we calculate the 5th, 35th, 65th, and 95th percentile of systems, which are the threshold values for this metric. We find these threshold values in Tables 8.29 and 8.30.

8.11 Conclusion

Above we have seen that not all metrics have a ready-made method for determining thresholds. The Alves and Baggen methods work in many cases, but we have shown that there are still problematic metrics where the methods do not work quite right.
The first and second questions are answered in the tables above. There we find the risk profiles and threshold values for the metrics that were analyzed.
To answer the third question, we have seen that for metrics with low variability, for example with many 0 values, it is difficult to choose the appropriate threshold percentiles. In addition, we have also seen that the Alves method does not work on metrics whose high values are desired. We solved this by determining the system level thresholds with a modified Baggen method and skip the risk profile creation with the Alves method.

8.11.1 Threats to validity

Our choice to include in certain sections only the files that were actually tested may be a threat to validity. Thus, we did not do our analysis on all code, but on a selection of it. This of course affects the results, as we have seen above. This is a form of selection bias that we will elaborate on next.
Selection bias is also a threat to validity in this part of our experiment. The results of this analysis is only applicable over this corpus, including all modifications and filtering we did, so we do not know how accurately it assesses systems from other corpora. We will discuss this in more detail in the validation chapter.
Finally, we cannot establish with certainty that the modifications done to estimate threshold metrics using the Alves method [9] and Baggen method [10] are actually correct. The methods do not describe how to handle metrics where larger values are desired and metrics that we can only measure at the system level. Applying these methods to these metrics requires modifications, as described above. We have based the modifications on the Maintainability model [8], which has handled metrics that are only measured at the system level in the same way. The paper describing this model does not address this or justify the correctness of these modifications. This threatens the validity of the method used to yield the results mentioned in this section.

Chapter 9

Validation

In this section, we will challenge the reliability of our results obtained from our experiment. We do this for each of our experiments described in Chapters 6, 7, and 8.

9.1 Comparative Analysis

In our comparative analysis, we had to deal with selection bias. We performed our experiment on a single Java and CSharp corpus. Thus, we do not know whether we would have obtained the same results if we had chosen a different corpus.
To expose the influence of this selection bias, we used bootstrapping. This involves running the experiment a number of times, and randomly selecting 50% of our corpus as the dataset. A large amount of variance and different results between the iterations and our experiments indicate a high influence of the selected corpus on our results. The bootstrapping results can be found in Appendix A. For evaluating the variance, we look specifically at the 70th, 80th, and 90th percentiles because these are the percentiles that are usually appropriate for determining thresholds.

Java and CSharp production code

We see in the tables B.8, B.17 and B.26 a low variance for the unit-level metrics.
At the file level, we also observe a low variance in the differences between Java and CSharp code between the different iterations. These can be found in the tables B.35, B.44, B.54, B.62 and B.71. At the system level, we do see a large variance in almost all metrics, allowing us to conclude that the selection of our corpus likely had a large impact on our system-level results. This variance can be observed in the tables B.80, B.89, B.98, B.107 and B.116.

In the same way as above, for the following categories: Java and CSharp test code, Java Production and test code and CSharp production and test code, we will look at the variance exhibited in the respective tables in Appendix A.

Java and CSharp test code

At the unit level, we see low variance in the tables for each metric. At the file level, we see higher variance for assert metrics at the 80th and 90th percentiles, for example. At the system level, we see very high variance for almost all metrics.

Java production and test code / CSharp production and test code

For the categories Java production and test code and CSharp production and test code, we see the same pattern as we have seen in the categories above. At the unit level, we see little variance. At file level it is a lot more and at system level it is clearly visible that the variance is very high.

9.1.1 Conclusion

The variance is generally low at the unit level, moderate at the file level, and high at the system level. One possible cause is that we have many Unit data points, fewer file data points and even fewer system

data points. Perhaps this means that our data set is too small, so the random selection of data points causes a lot of variance.
From this we can conclude that our experiment is reliable at unit level, but not at file and system level. In the future, our experiment could be repeated with a larger corpus of Java and CSharp projects, with more File and System data points. This might produce more reliable results than the results in the experiments above.

9.2 Correlation Analysis

Also for the correlation study, we applied bootstrapping to see how large the variance is between different iterations of our experiment, conducted on a portion of our corpus. When this variance is large, we can conclude that the results of our experiment are not reliable and that selection bias is high. In that case, the selection of our dataset has a great influence on the results obtained. The results of this bootstrapping analysis can be found in Appendix C.

Java

Since we saw above that there was a large variance at the system level, and a moderate variance is at the file level we expect to see the same thing reflected in our correlation validation. This is because we suspect that our file and system datasets are too small, which causes a large variance. We then expect this to translate into a higher variance at the file and system level for the correlations.
For Java, for all unit-level metrics, we see that the variance is low, and all iterations are close to the mean. So from this we cannot conclude that our results are unreliable. Again, at a file level, the variance is negligible and the correlations are all close to the mean. We see a somewhat larger variance on a system level, but it is still all very close to the mean and not enough to draw the conclusion that our results are unreliable.

CSharp

For CSharp, we expect the same outcomes as for Java. We see that at the unit level the variance is low for all correlations. Also, at the file level, we see hardly any variation between the correlations and cannot conclude that our results are unreliable. At the system level, we observe the largest variance, but it too is negligible and does not cause us to draw different conclusions from the results of the individual iterations.

9.2.1 Conclusion

For the correlations, we see values at all levels that are very close to the mean and little different from each other. We see the greatest variance at the system level, but even at this level the variance is low. As a result, we conclude that the variance at all levels is too low to show a high influence of selection bias in our results.

9.3 Threshold study

To verify the results and performance of our threshold study, we will test 6 projects tested with SonarQube. The projects were chosen so that 3 projects were rated as poor by SonarQube and 3 were rated as good. We will verify that our model, with the calculated thresholds, draws the same conclusions about these projects.

9.3.1 Projects

The projects were chosen based on the scores calculated by SonarQube that are on SonarCloud. A limited number of metrics calculated across projects are shown here. Because of this small number of metrics, and the small number of test metrics that SonarQube calculates in general, it is difficult to properly distinguish between well and poorly tested projects. For the selection, we looked at the number of bugs, code smells, the percentage of code coverage, and the number of unit tests. When rated A by SonarQube, these scores are good according to SonarQube. For the poorly tested projects, we have selected projects that score less well on these metrics.

We chose these metrics because we believe they are direct or indirect indicators of test quality. Code coverage is one of the metrics that is in the SIG Test code quality model. We use the number of unit tests as a proxy for the number of test code LOC, although this is unlikely to be very accurate. The number of code smells and bugs are indirect indicators of poor test code, since writing a good test suite should result in a lower number of code smells and bugs.

	Code Coverage	Unit tests	Bugs	Code smells
BxBot	91%	543	0	34
Trellis	99.5%	1800	0	255
Apache RNG	99.7%	2500	0	39

Table 9.1: Metrics for well tested projects.

	Code Coverage	Unit tests	Bugs	Code smells
WebGoat	31%	236	41	731
Replicator	20.6%	85	31	763
Service-web	14.2%	409	11	590

Table 9.2: Metrics for not well tested projects.

9.3.2 Metrics

We chose to validate only the Assert/CC, Assert/Branchpoint, and Test/production code ratio metrics, since we concluded that the Unverified CC and unverified Branchpoint metrics are both highly correlated with other metrics and thus add little value to the model.

	LOC	CC	Branchpoints	Direct_Asserts	LOC_test	Assert/branch	Assert/CC	Test/prod
bxbot	9959	1536.0	519.0	0.0	13911	0.000000	0.000000	1.396827
rng	2889	527.0	113.0	97.0	5074	0.858407	0.184061	1.756317
trellis	10535	1806.0	692.0	1735.0	20335	2.507225	0.960687	1.930233
replicator	8961	1663.0	749.0	162.0	3587	0.216288	0.097414	0.400290
service_web	11600	2200.0	1016.0	276.0	3562	0.271654	0.125455	0.307069
webgoat	9427	1648.0	607.0	41.0	5468	0.067545	0.024879	0.580036

Table 9.3: Metrics for the validation projects.

9.3.3 Star scores

	Assert/branch	Assert/CC	Test/prod
BxBot	⋆	⋆	⋆⋆
Trellis	⋆⋆⋆	⋆⋆⋆	⋆⋆⋆
Apache RNG	⋆⋆⋆⋆	⋆⋆⋆⋆⋆	⋆⋆⋆
WebGoat	⋆⋆	⋆⋆⋆	⋆
Replicator	⋆⋆⋆	⋆⋆⋆	⋆
Service-web	⋆⋆	⋆⋆	⋆

Table 9.4: Star scores for good and bad systems.

In Table 9.4, we see the star scores of the systems for each metric. The first three projects are the well-tested projects. The last three are less well tested. Based on the star scores obtained, we will evaluate how good our thresholds are in determining test code quality, for these metrics.

What immediately stands out are the low metrics that the BxBot project exhibits for the Assert/branch and Assert/CC metrics. The reason for this is that the number of Direct Asserts for this project is 0. On the other hand, we can observe that this project has 543 unit tests. This means that this project did not follow the naming convention so the algorithm that calculates direct asserts did not find any. Despite having adequate unit tests for an Assert/Branch metric of 0.5, the project scores poorly on these two metrics.
The star scores are, on average, substantially closer together than Tables 9.1 and 9.2 suggest. We can see that the star scores of the Trellis and Replicator projects for the Assert/branch and Assert/CC metrics are the same, yet the number of unit tests and test coverage of these metrics are significantly different. As a result, these metrics are unable to capture this disparity and translate it into star ratings.

We conclude that the good results of the first three projects and the poor results of the last three projects on SonarQube do not translate into our star ratings. This means that the metrics cannot very accurately differentiate between good and bad tested projects. We conclude this by the low scores that the BxBot project shows, and the small difference between the Replicator and Trellis projects which gives the impression that these two projects have similar test code quality.

Chapter 10

Conclusion

In this research, we focused on three sub components that together should lead to an improvement of the SIG Test code quality model and provide new insights into determining test code quality and calculating thresholds of software metrics. We initially looked at whether we needed to separate the dataset by programming language and test/production context for our threshold derivation study. We concluded that the accuracy of the threshold values is improved by separating the data points based on programming language and context.
Next we put the various measures to the test by comparing how closely they correlated with one another and with simple metrics to see which ones are redundant and don't bring value to the model. As a result, we were left with a list of metrics that we couldn't prove were duplicated. These are the code metrics: Assert/Branch, Assert/CC, and Test/Production.
Finally, using the results of the two previous conclusions, we computed the threshold value. We were able to experience the use of multiple threshold benchmark methods and discover the pain points in their use in addition to computing the thresholds. For example, we've seen that the approaches might not be suitable for all types of metrics, particularly metrics where high values are desired and ratio metrics where a denominator of 0 is a possibility.

10.1 Future work

First, we believe our work can be repeated using a different dataset to determine if the results are the same. With our validation study for our comparison analysis, we were able to determine that our results are not accurate at the file and system level due to selection bias. Another explanation could be that there aren't enough data points at the file and system level. An interesting follow-up study would be to replicate the study with a different corpus with more file and system level data points.
Our comparative study can be drawn more broadly than what we have focused on. It can focus on how different code types and programming languages differ from each other and how this looks like in real code. These differences can then form the basis for choices to be made such as whether to combine code types in a corpus for specific research.
Our correlation study could be extended to include alternative metrics from the literature that may be easier to comprehend for the end user or easier to calculate. These could then be used as a proxy for one of the metrics we propose as candidates for the SIG Test code quality model.
When we used the Baggen and Alves methodologies for our metrics during our threshold study, we ran into several issues. A follow-up study could be conducted to confirm the divergent use of these methodologies, as we have seen in this study, or to develop a new methodology that is more suited to these problematic metrics, such as those that desire high values or ratio metrics with a denominator of 0.

Bibliography

[1] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering (FOSE '07)*, 2007, pp. 85–103. DOI: 10.1109/FOSE.2007.25.

[2] M. Ellims, J. Bridges, and D. Ince, "The economics of unit testing," *Empirical Software Engineering*, vol. 11, Mar. 2006. DOI: 10.1007/s10664-006-5964-9.

[3] G. Meszaros, *xUnit Test Patterns: Refactoring Test Code*, ser. Addison-Wesley Signature Series. Addison-Wesley, 2007, ISBN: 978-0-13-149505-0. [Online]. Available: https://www.safaribooksonline.com/library/view/xunit-test-patterns/9780131495050/.

[4] K. Beck, *Test Driven Development. By Example (Addison-Wesley Signature)*. Addison-Wesley Longman, Amsterdam, 2002, ISBN: 0321146530.

[5] S. D. P. F. Z. A. B. M. B. Alberto, "On the relation of test smells to software code quality," 2018. DOI: https://doi.org/10.1109/ICSME.2018.00010.

[6] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman, "Test code quality and its relation to issue handling performance," *IEEE Transactions on Software Engineering*, vol. 40, pp. 1100–1125, 2014.

[7] P. van Beckhoven, "Assessing test suite effectiveness using static analysis," 2017.

[8] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," Oct. 2007, pp. 30–39, ISBN: 978-0-7695-2948-6. DOI: 10.1109/QUATIC.2007.8.

[9] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," *2010 IEEE International Conference on Software Maintenance*, pp. 1–10, 2010.

[10] R. Baggen, J. P. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Quality Journal*, vol. 20, pp. 1–21, Jun. 2011. DOI: 10.1007/s11219-011-9144-9.

[11] K. S. University, *Spss tutorials: Pearson correlation*, https://libguides.library.kent.edu/SPSS, 2021.

[12] J. Visser, "Building maintainable software," *Software Improvement Group, B.V.*, vol. First Release, 2016.

[13] Oracle, "Programming with assertions," 2021.

[14] P. Susan DeanBarbara Illowsky, "Descriptive statistics: Skewness and the mean, median, and mode.," 2012. [Online]. Available: http://cnx.org/contents/6c4fb0df-8562-40db-8e48-3d91d7dd65f7@9.

[15] W. P. H., "Kurtosis as peakedness," 2014.

[16] Wikipedia, *File:Pearson type VII distribution PDF.svg — Wikipedia, the free encyclopedia*, https://commons.wikimedia.org/wiki/File:Pearson_type_VII_distribution_PDF.svg, [Online; accessed 19-August-2021], 2020.

[17] P. S. Foundation, *Python programming languague*, https://www.python.org/, 2021.

[18] PyData, *Pandas (python library)*, https://pandas.pydata.org/, 2021.

[19] D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju, "Empirical analysis of the relationship between cc and sloc in a large corpus of java methods and c functions," *Journal of Software: Evolution and Process*, vol. 28, no. 7, pp. 589–618, 2016. DOI: https://doi.org/10.1002/smr.1760. eprint: https://www.onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1760. [Online]. Available: https://www.onlinelibrary.wiley.com/doi/abs/10.1002/smr.1760.

[20] djechelon, *Spring-security*, https://github.com/spring-projects/spring-security/blob/main/core/src/main/java/org/springframework/security/core/userdetails/UserDetails.java, 2021.

[21] Oracle, "The java™ tutorials - defining an interface," 2021.

[22] djechelon, *Spring-security*, https://github.com/spring-projects/spring-security/blob/main/core/src/main/java/org/springframework/security/authentication/dao/DaoAuthenticationProvid
java, 2021.

[23] ——, *Spring-security*, https://github.com/spring-projects/spring-security/blob/main/buildSrc/src/test/resources/samples/javadocapi/multimodule/api/src/main/java/sample/Api.java, 2021.

[24] A. C. Murthy, *Hadoop*, https://github.com/apache/hadoop/blob/trunk/hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager/src/main/java/org/apache/hadoop/yarn/server/nodemanager/webapp/AggregatedLogsPage.java, 2021.

[25] pniederw, *Gradle*, https://github.com/gradle/gradle/blob/master/subprojects/scala/src/integTest/resources/org/gradle/scala/compile/ZincScalaCompilerIntegrationTest/compilesJavaCodeIncrementally/src/main/scala/Other.java, 2021.

[26] J. sung Chung, *Tensorflow*, https://github.com/tensorflow/tensorflow, 2021.

[27] A. Pipatpinyopong, *Facebook-yoga*, https://github.com/facebook/yoga, 2021.

[28] George and Mallery, "Pss for windows step by step: A simple guide and reference," *Boston: Pearson.*, 2010.

Appendix A

Comparative analysis results

A.1 Java production - CSharp production

A.1.1 Unit level

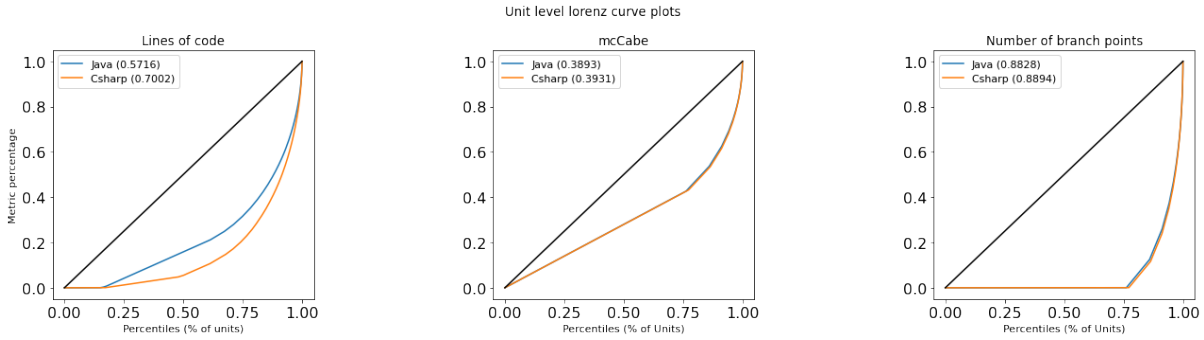

Figure A.1: Lorenz plots between Java and CSharp production units.

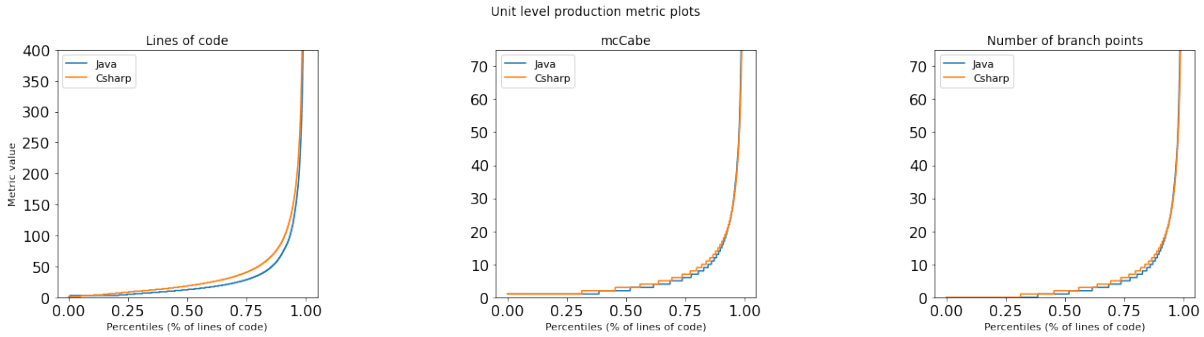

Figure A.2: Cumulative percentile plots between Java and CSharp production units.

Table A.1: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2238.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	28.57%	93.57%

Table A.2: Percentile values of Java production and CSharp production for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	39.69%

Table A.3: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	39.74%

A.1.2 File level

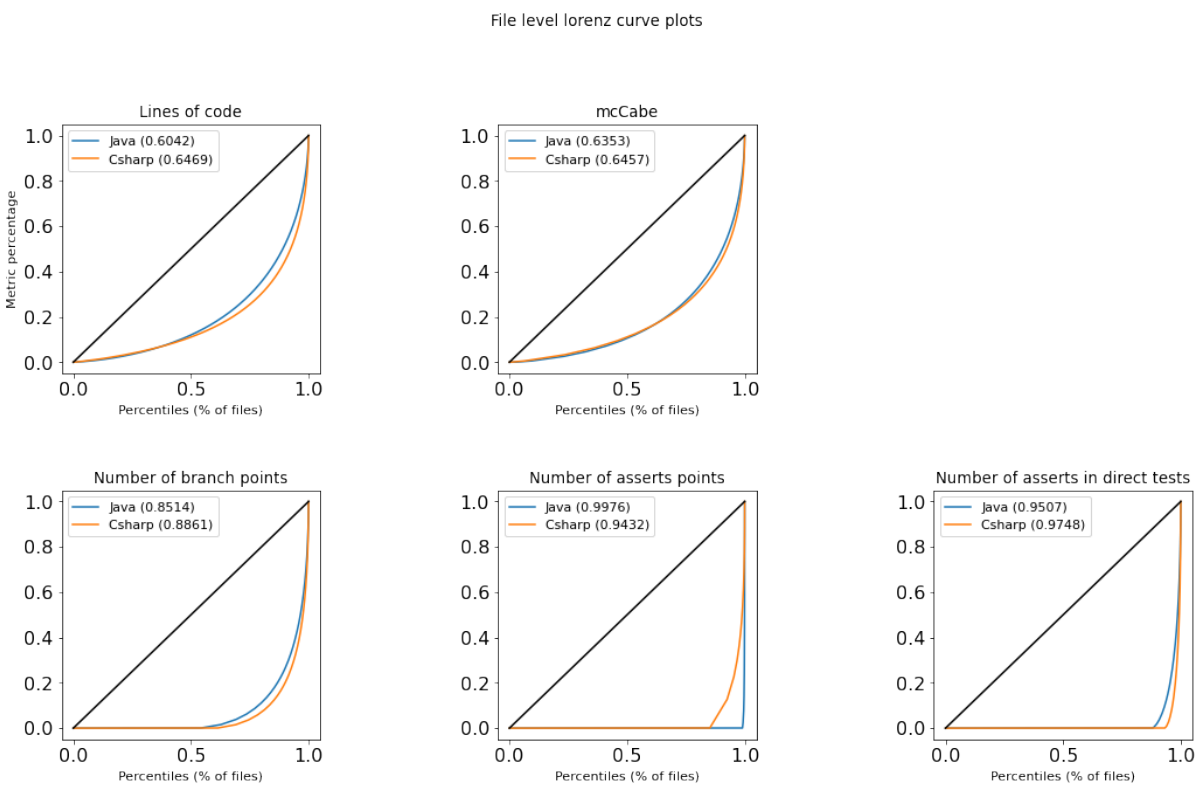

Figure A.3: Lorenz plots between Java and CSharp production files.

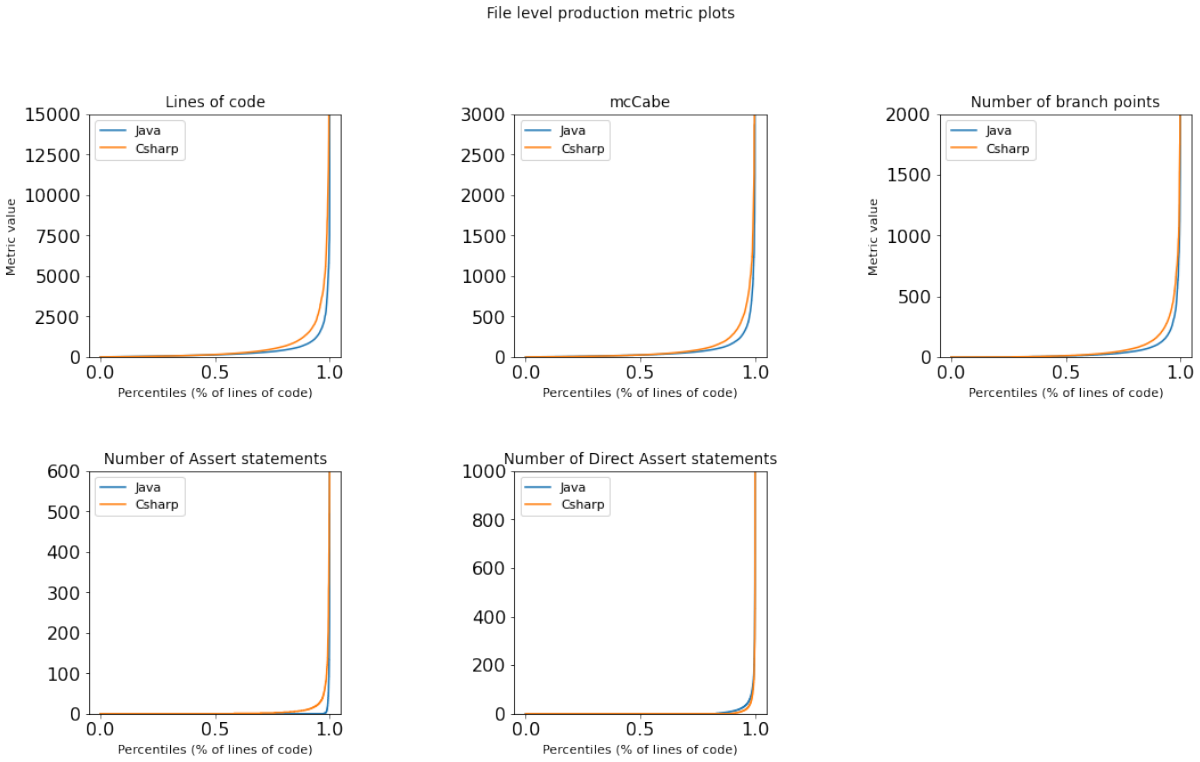

Figure A.4: Cumulative percentile plots between Java and CSharp production files.

Table A.4: Percentile values of Java production and CSharp production for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	197.0	283.0	434.0	805.0	9195.0
CSharp production SLOC	23.0	42.0	68.0	106.0	159.0	246.0	388.0	667.0	1417.0	36664.0
Differences	26.09%	19.05%	7.35%	2.91%	11.97%	24.87%	37.1%	53.69%	76.02%	298.74%

Table A.5: Percentile values of Java production and CSharp production for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	170.0	3402.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	42.0	69.0	121.0	272.0	5822.0
Differences	0.0%	14.29%	18.18%	0.0%	3.85%	13.51%	27.78%	40.7%	60.0%	71.13%

Table A.6: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	104.0	1963.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	74.0	171.0	4349.0
Differences	~	~	0.0%	20.0%	33.33%	31.25%	44.44%	51.02%	64.42%	121.55%

Table A.7: Percentile values of Java production and CSharp production for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	1149.0
Differences	~	~	~	~	~	~	~	~	~	38.6%

Table A.8: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	933.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2766.0
Differences	~	~	~	~	~	~	~	~	900.0%	196.46%

A.1.3 System level

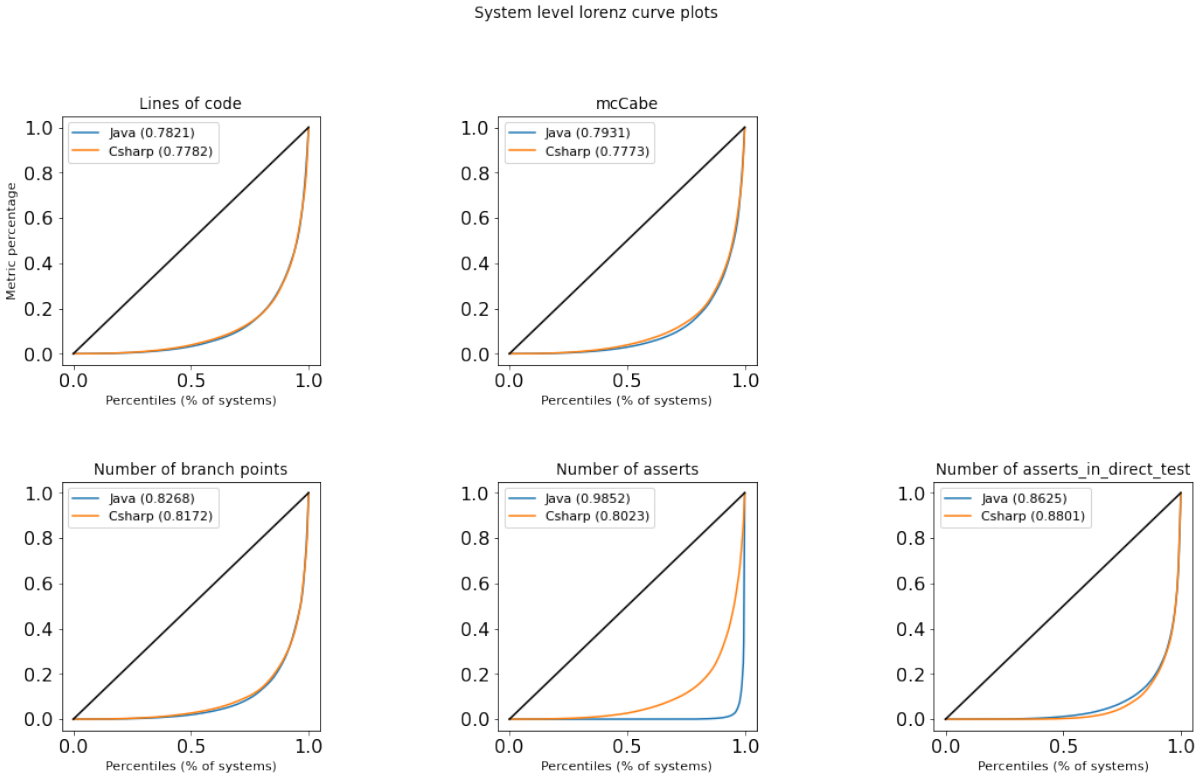

Figure A.5: Lorenz plots between Java and CSharp production system.

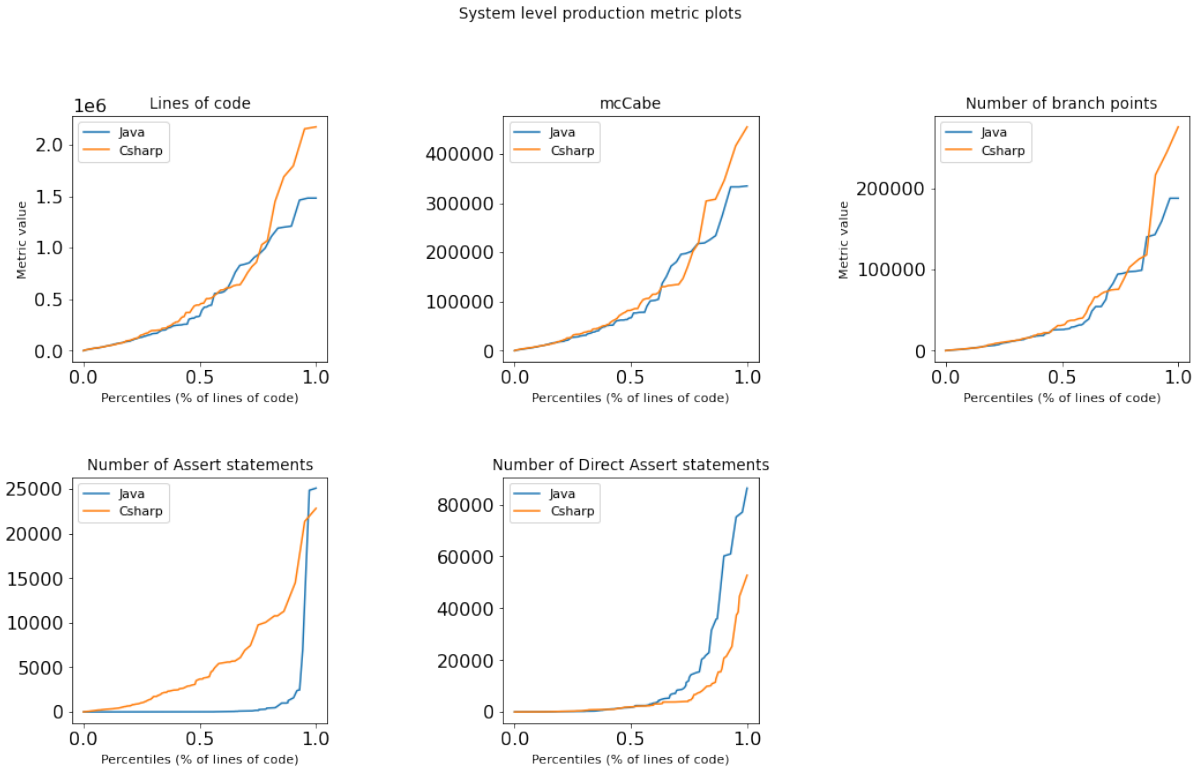

Figure A.6: Cumulative percentile plots between Java and CSharp production system.

Table A.9: Percentile values of Java production and CSharp production for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	43093.0	92674.0	163844.0	246038.0	335453.0	563063.0	839063.0	995766.0	1209227.0	1481603.0
CSharp production SLOC	44381.0	99098.0	194945.0	278093.0	442590.0	587560.0	691704.0	1069001.0	1689227.0	2154519.0
Differences	2.99%	6.93%	18.98%	13.03%	31.94%	4.35%	21.3%	7.35%	39.69%	45.42%

Table A.10: Percentile values of Java production and CSharp production for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7899.0	18359.0	31072.0	50813.0	66128.0	101506.0	180293.0	217534.0	275706.0	334705.0
CSharp production CC	8582.0	19168.0	37055.0	52958.0	81631.0	114470.0	133994.0	218716.0	307757.0	454793.0
Differences	8.65%	4.41%	19.26%	4.22%	23.44%	12.77%	34.55%	0.54%	11.63%	35.88%

Table A.11: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2414.0	6268.0	12021.0	18061.0	25915.0	35034.0	73326.0	97166.0	139875.0	187433.0
CSharp production Branchpoints	2800.0	7958.0	12701.0	20300.0	30834.0	40252.0	72772.0	102699.0	117813.0	243970.0
Differences	15.99%	26.96%	5.66%	12.4%	18.98%	14.89%	0.76%	5.69%	18.73%	30.16%

Table A.12: Percentile values of Java production and CSharp production for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	8.0	30.0	100.0	421.0	1264.0	25081.0
CSharp production asserts	301.0	686.0	1676.0	2455.0	3567.0	5479.0	6907.0	10041.0	11276.0	21348.0
Differences	~	~	83700.0%	81733.33%	44487.5%	18163.33%	6807.0%	2285.04%	792.09%	17.49%

Table A.13: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	19.0	169.0	205.0	1031.0	1801.0	3435.0	8323.0	15473.0	35964.0	86309.0
CSharp production Direct asserts	8.0	179.0	526.0	914.0	1875.0	2863.0	3830.0	7469.0	16522.0	52714.0
Differences	137.5%	5.92%	156.59%	12.8%	4.11%	19.98%	117.31%	107.16%	117.67%	63.73%

A.2 Java test - CSharp test

A.2.1 Unit level

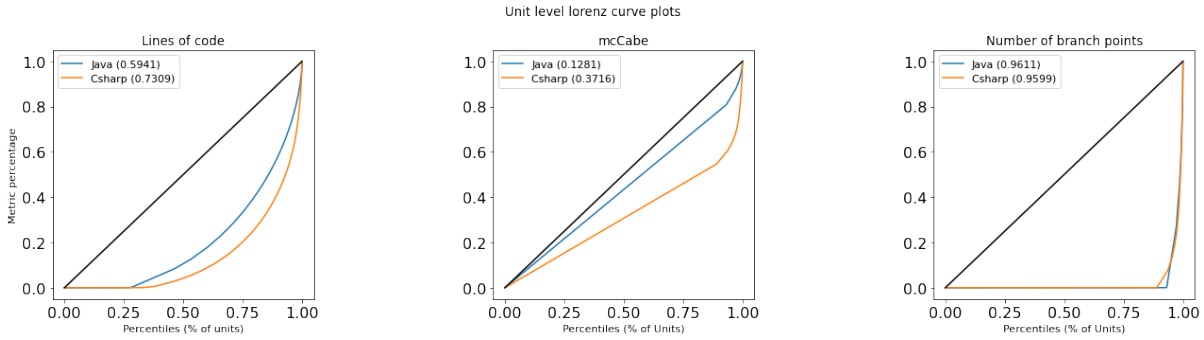

Figure A.7: Lorenz plots between Java and CSharp test units.

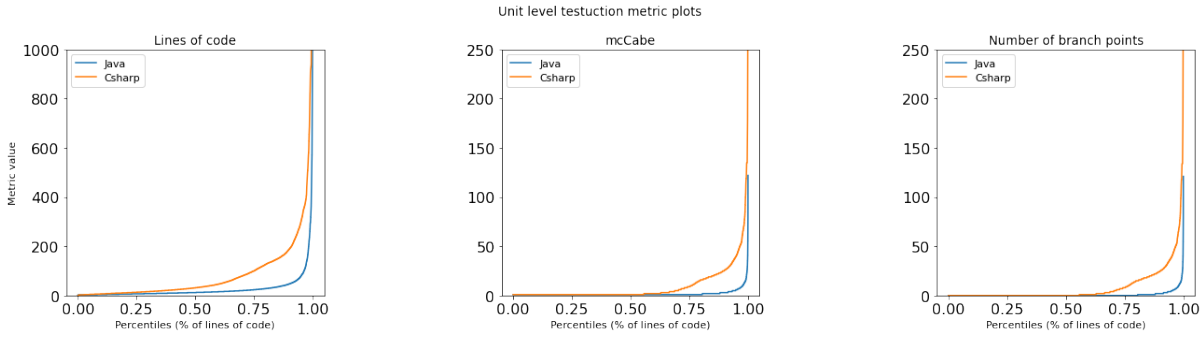

Figure A.8: Cumulative percentile plots between Java and CSharp test units.

Table A.14: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	23.0	31.0	47.0	80.0	126.0	188.0	3605.0
Differences	75.0%	83.33%	100.0%	130.0%	158.33%	193.75%	300.0%	350.0%	300.0%	126.44%

Table A.15: Percentile values of Java test and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	193.44%

Table A.16: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	~	~	~	1150.0%	195.04%

A.2.2 File level

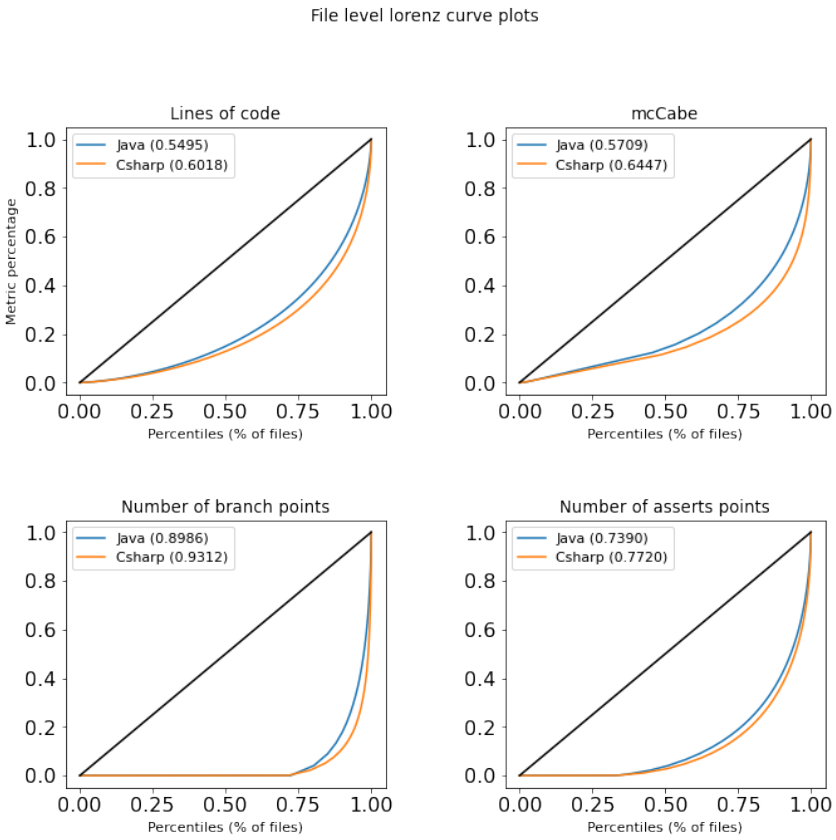

Figure A.9: Lorenz plots between Java and CSharp test files.

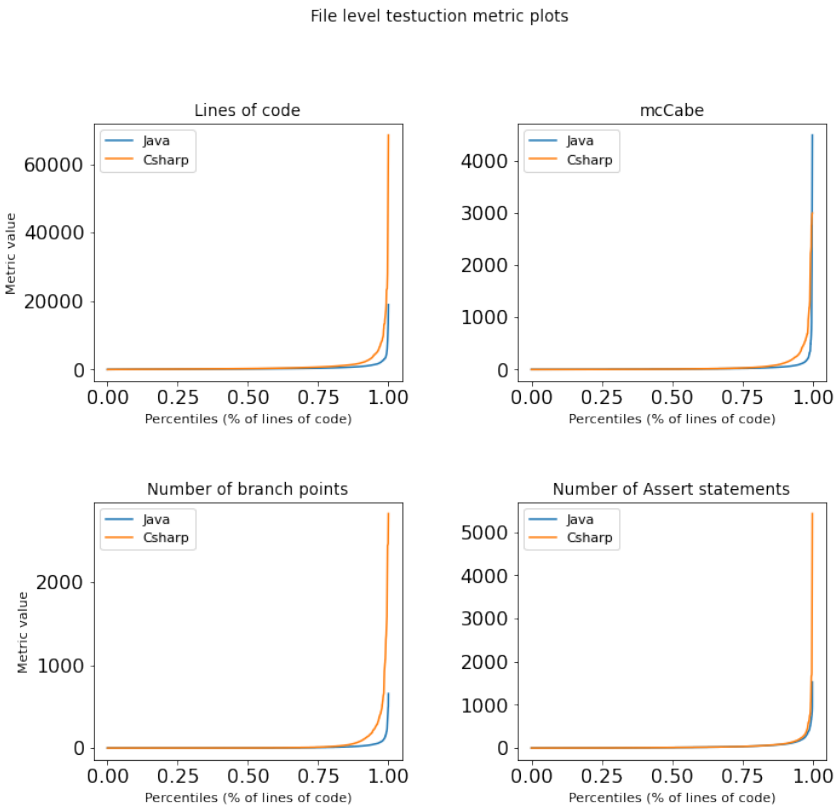

Figure A.10: Cumulative percentile plots between Java and CSharp test files.

Table A.17: Percentile values of Java test and CSharp test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	239.0	330.0	484.0	855.0	18933.0
CSharp test SLOC	53.0	86.0	126.0	178.0	253.0	367.0	558.0	901.0	1854.0	68502.0
Differences	15.22%	21.13%	27.27%	33.83%	42.94%	53.56%	69.09%	86.16%	116.84%	261.81%

Table A.18: Percentile values of Java test and CSharp test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	4488.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	42.0	123.0	2972.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	9.09%	23.53%	55.56%	136.54%	51.01%

Table A.19: Percentile values of Java test and CSharp test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	656.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	82.0	2828.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	256.52%	331.1%

Table A.20: Percentile values of Java test and CSharp test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	1529.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	17.0	27.0	47.0	98.0	4498.0
Differences	~	0.0%	33.33%	0.0%	0.0%	0.0%	0.0%	4.44%	10.11%	194.18%

A.2.3 System level

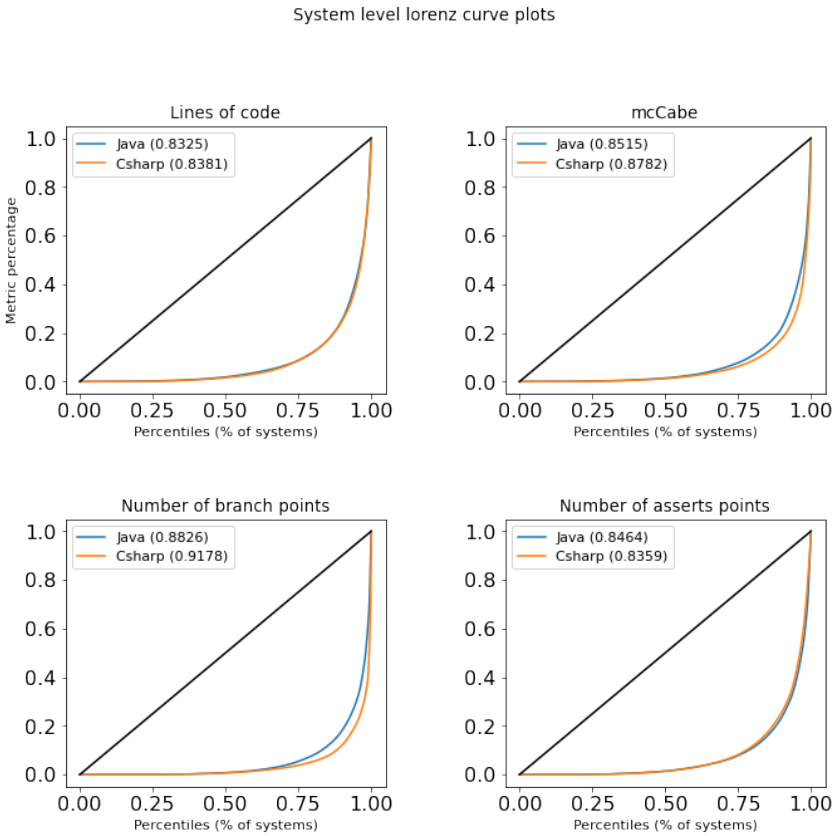

Figure A.11: Lorenz plots between Java and CSharp test system.

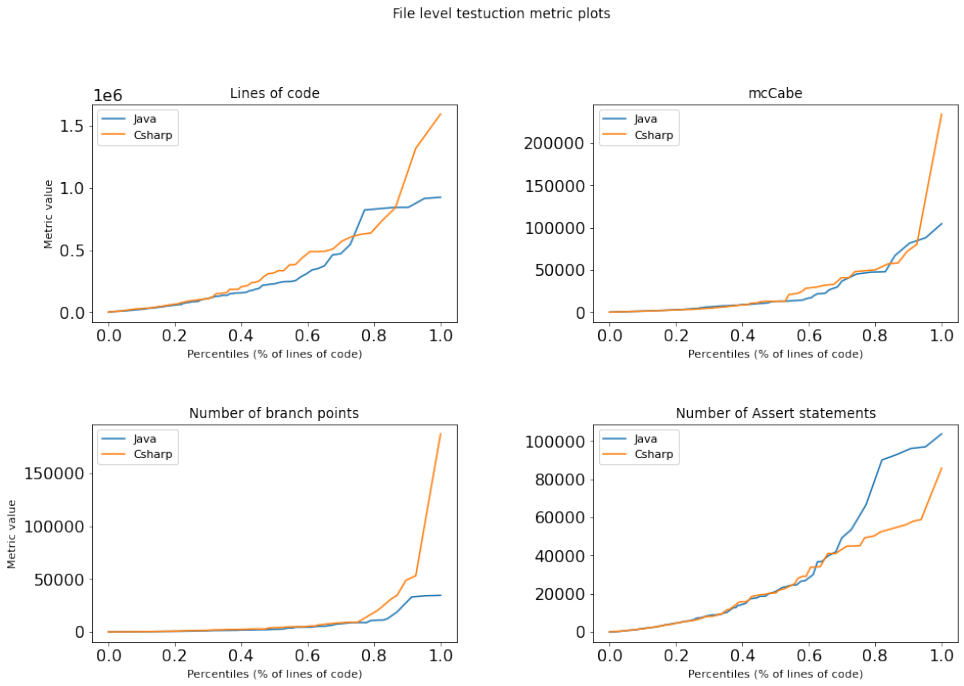

Figure A.12: Cumulative percentile plots between Java and CSharp test system.

Table A.21: Percentile values of Java test and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	22930.0	56141.0	106772.0	155106.0	227658.0	306151.0	470474.0	822316.0	843011.0	915395.0
CSharp test SLOC	28175.0	64248.0	108472.0	204276.0	313855.0	437622.0	508226.0	636785.0	840121.0	1594573.0
Differences	22.87%	14.44%	1.59%	31.7%	37.86%	42.94%	8.02%	29.14%	0.34%	74.2%

Table A.22: Percentile values of Java test and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1142.0	2749.0	6033.0	8748.0	12592.0	15173.0	36566.0	47309.0	66771.0	87887.0
CSharp test CC	1146.0	2592.0	4569.0	8364.0	12847.0	28037.0	40618.0	47865.0	71616.0	234164.0
Differences	0.35%	6.06%	32.04%	4.59%	2.03%	84.78%	11.08%	1.18%	7.26%	166.44%

Table A.23: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	245.0	563.0	1133.0	1780.0	2465.0	4401.0	7256.0	10829.0	18823.0	34532.0
CSharp test Branchpoints	256.0	760.0	1486.0	2359.0	3843.0	4963.0	8125.0	9188.0	48751.0	186983.0
Differences	4.49%	34.99%	31.16%	32.53%	55.9%	12.77%	11.98%	17.86%	159.0%	441.48%

Table A.24: Percentile values of Java test and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1736.0	4697.0	8632.0	14063.0	21127.0	26800.0	49274.0	66636.0	92921.0	96968.0
CSharp test asserts	1609.0	4538.0	8011.0	15738.0	20400.0	29171.0	41071.0	50276.0	56122.0	85742.0
Differences	7.89%	3.5%	7.75%	11.91%	3.56%	8.85%	19.97%	32.54%	65.57%	13.09%

A.3 Java production - Java test

A.3.1 Unit level

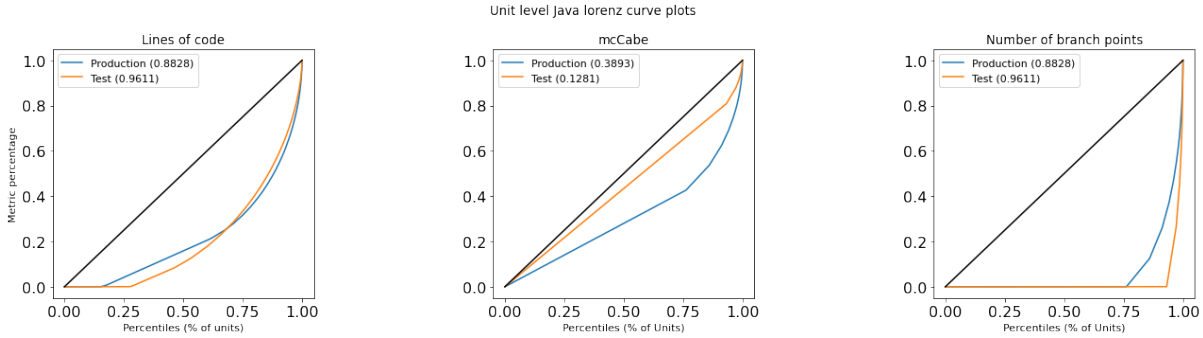

Figure A.13: Lorenz plots between Java production and Java test production units.

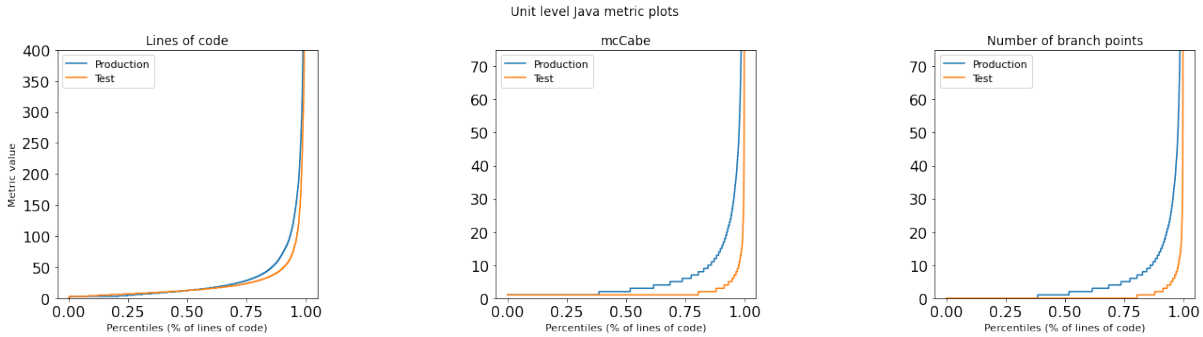

Figure A.14: Lorenz plots between Java production and Java test production units.

Table A.25: Percentile values of Java production and Java test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2238.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	48.94%	40.58%

Table A.26: Percentile values of Java production and Java test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	536.07%

Table A.27: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
Differences	~	~	~	~	~	~	~	~	600.0%	540.5%

A.3.2 File level

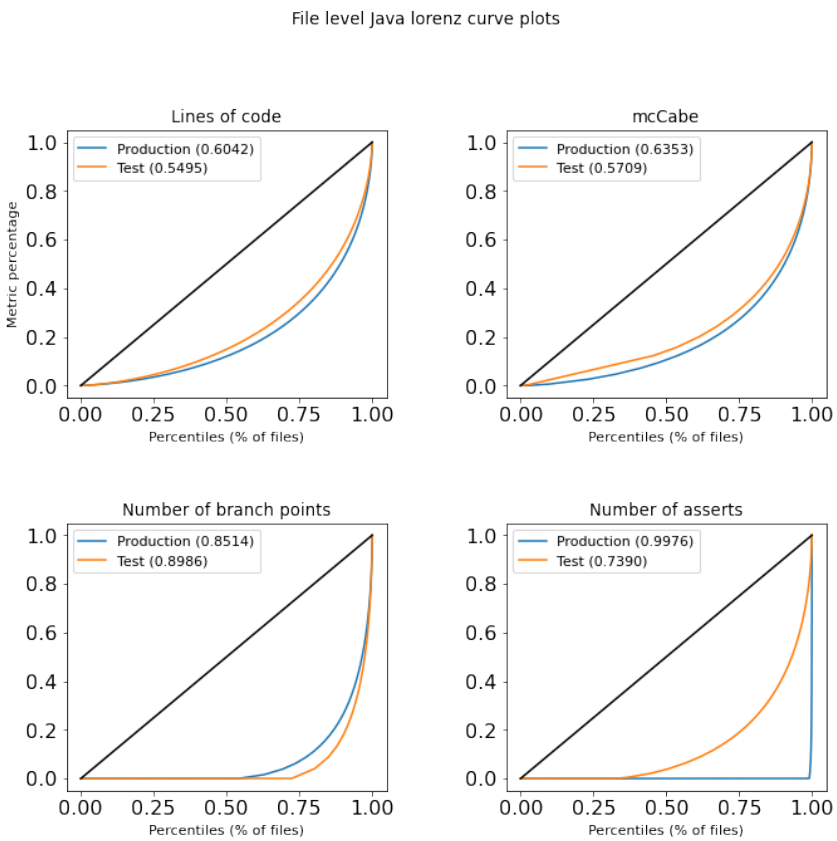

Figure A.15: Lorenz plots between Java production and Java test production files.

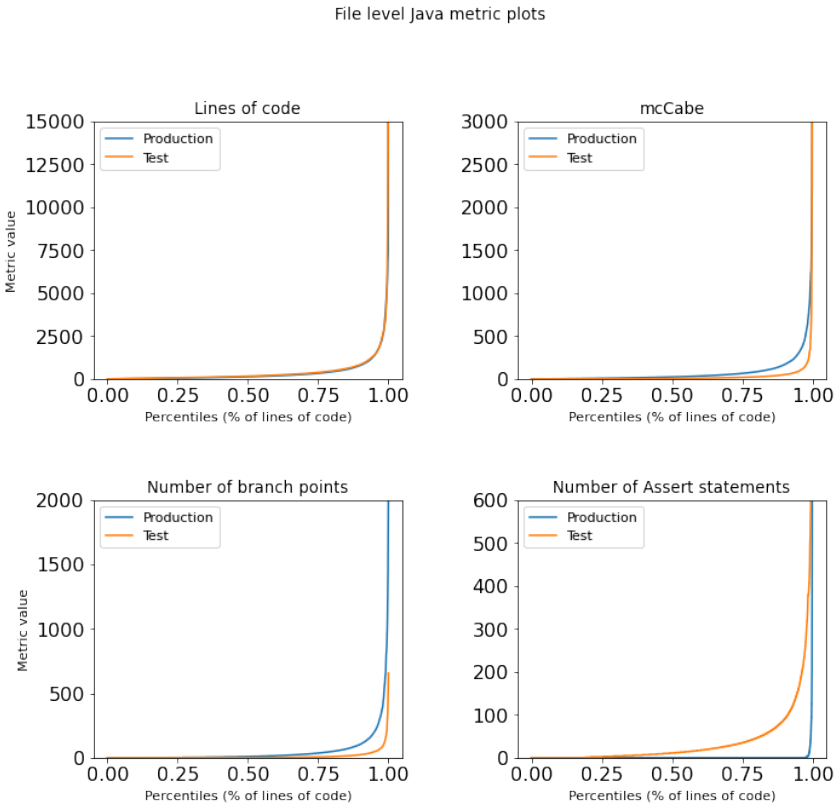

Figure A.16: Lorenz plots between Java production and Java test production files.

Table A.28: Percentile values of Java production and Java test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	197.0	283.0	434.0	805.0	9195.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	239.0	330.0	484.0	855.0	18933.0
Differences	58.62%	42.0%	35.62%	29.13%	24.65%	21.32%	16.61%	11.52%	6.21%	105.91%

Table A.29: Percentile values of Java production and Java test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	170.0	3402.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	4488.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	218.52%	226.92%	31.92%

Table A.30: Percentile values of Java production and Java test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	104.0	1963.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	656.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	352.17%	199.24%

Table A.31: Percentile values of Java production and Java test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	1529.0
Differences	~	~	~	~	~	~	~	~	~	84.44%

A.3.3 System level

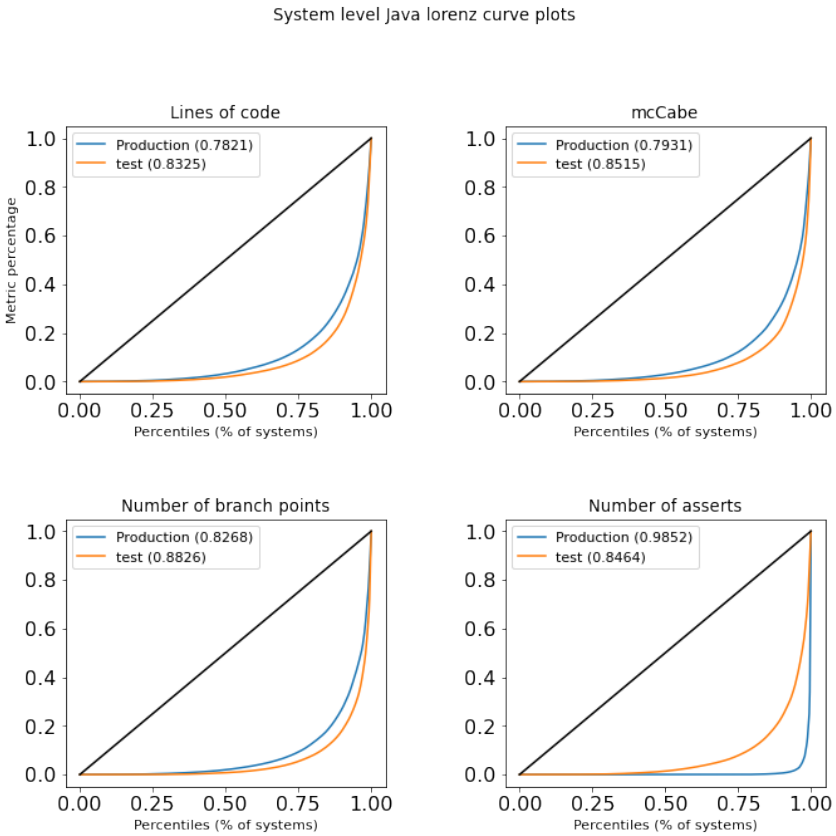

Figure A.17: Lorenz plots between Java production and Java test production systems.

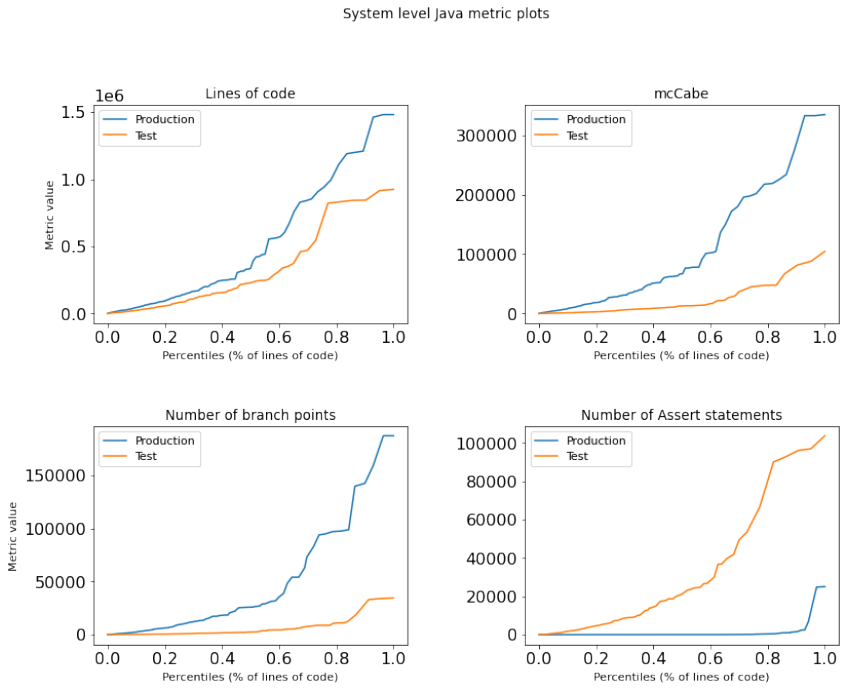

Figure A.18: Lorenz plots between Java production and Java test production systems.

Table A.32: Percentile values of Java production and Java test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	43093.0	92674.0	163844.0	246038.0	335453.0	563063.0	839063.0	995766.0	1209227.0	1481603.0
Java test SLOC	22930.0	56141.0	106772.0	155106.0	227658.0	306151.0	470474.0	822316.0	843011.0	915395.0
Differences	87.93%	65.07%	53.45%	58.63%	47.35%	83.92%	78.34%	21.09%	43.44%	61.85%

Table A.33: Percentile values of Java production and Java test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7899.0	18359.0	31072.0	50813.0	66128.0	101506.0	180293.0	217534.0	275706.0	334705.0
Java test CC	1142.0	2749.0	6033.0	8748.0	12592.0	15173.0	36566.0	47309.0	66771.0	87887.0
Differences	591.68%	567.84%	415.03%	480.85%	425.16%	568.99%	393.06%	359.82%	312.91%	280.84%

Table A.34: Percentile values of Java production and Java test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2414.0	6268.0	12021.0	18061.0	25915.0	35034.0	73326.0	97166.0	139875.0	187433.0
Java test Branchpoints	245.0	563.0	1133.0	1780.0	2465.0	4401.0	7256.0	10829.0	18823.0	34532.0
Differences	885.31%	1013.32%	960.99%	914.66%	951.32%	696.05%	910.56%	797.28%	643.11%	442.78%

Table A.35: Percentile values of Java production and Java test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	8.0	30.0	100.0	421.0	1264.0	25081.0
Java test asserts	1736.0	4697.0	8632.0	14063.0	21127.0	26800.0	49274.0	66636.0	92921.0	96968.0
Differences	~	~	431500.0%	468666.67%	263987.5%	89233.33%	49174.0%	15728.03%	7251.34%	286.62%

A.4 CSharp production - CSharp test

A.4.1 Unit level

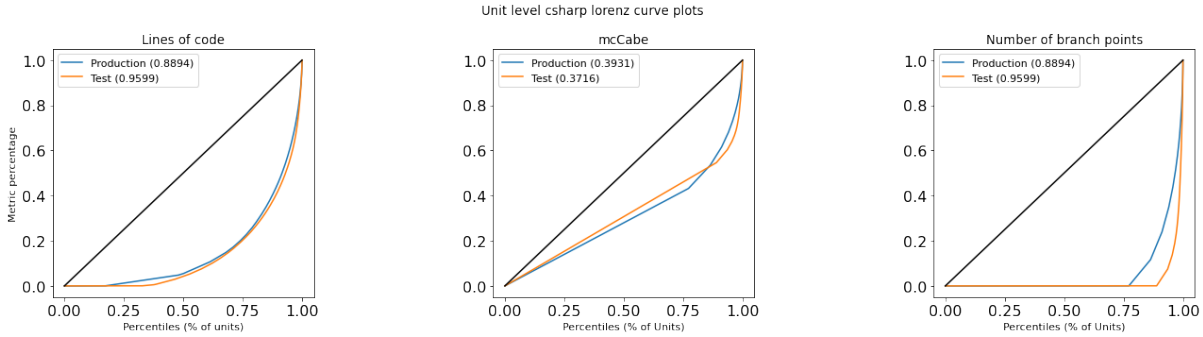

Figure A.19: Lorenz plots between CSharp production and CSharp test production units.

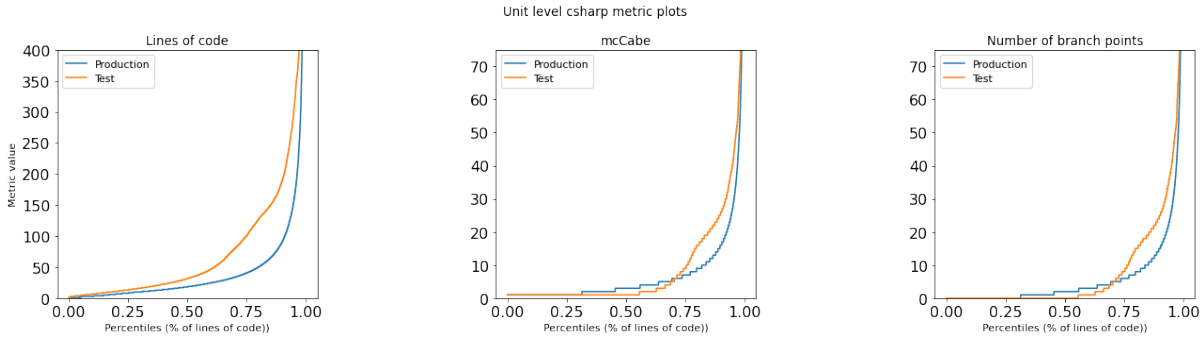

Figure A.20: Lorenz plots between CSharp production and CSharp test production units.

Table A.36: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	23.0	31.0	47.0	80.0	126.0	188.0	3605.0
Differences	133.33%	57.14%	60.0%	64.29%	72.22%	88.0%	135.29%	152.0%	108.89%	20.17%

Table A.37: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	202.79%

Table A.38: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	66.67%	203.36%

A.4.2 File level

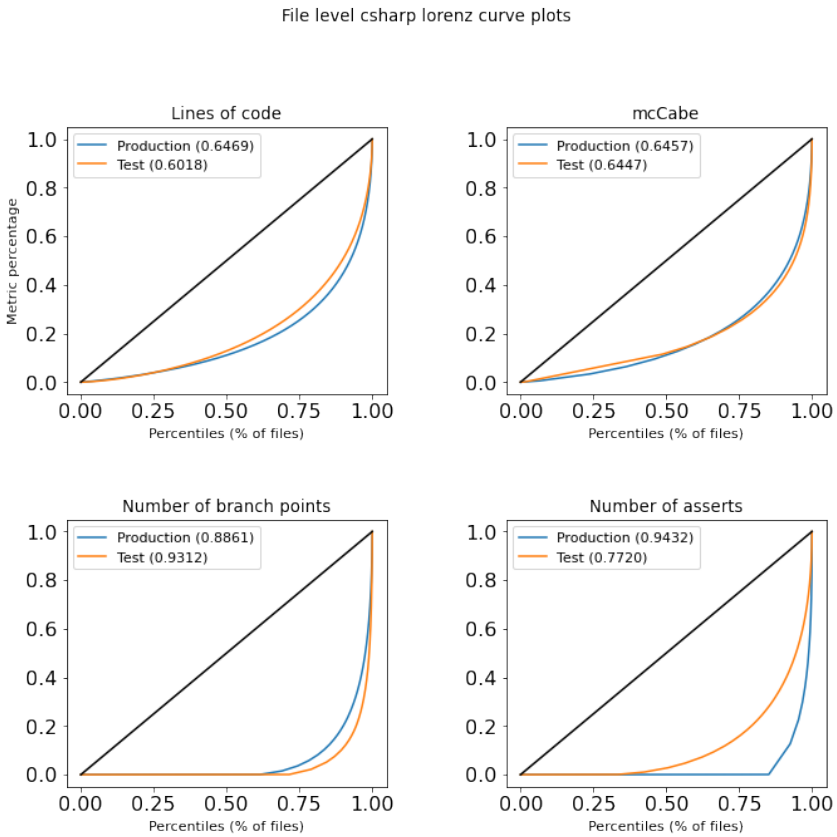

Figure A.21: Lorenz plots between CSharp production and CSharp test production Files.

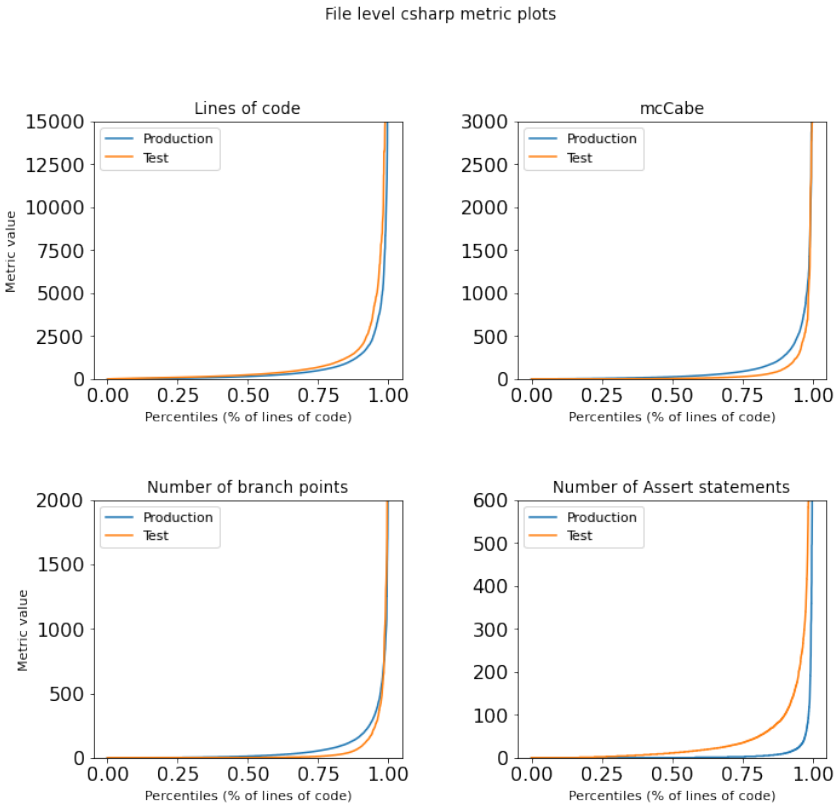

Figure A.22: Lorenz plots between CSharp production and CSharp test production Files.

Table A.39: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	68.0	106.0	159.0	246.0	388.0	667.0	1417.0	36664.0
CSharp test SLOC	53.0	86.0	126.0	178.0	253.0	367.0	558.0	901.0	1854.0	68502.0
Differences	130.43%	104.76%	85.29%	67.92%	59.12%	49.19%	43.81%	35.08%	30.84%	86.84%

Table A.40: Percentile values of CSharp production and CSharp test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	42.0	69.0	121.0	272.0	5822.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	42.0	123.0	2972.0
Differences	100.0%	250.0%	266.67%	260.0%	285.71%	250.0%	228.57%	188.1%	121.14%	95.9%

Table A.41: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	74.0	171.0	4349.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	82.0	2828.0
Differences	~	~	~	~	1100.0%	950.0%	550.0%	289.47%	108.54%	53.78%

Table A.42: Percentile values of CSharp production and CSharp test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	1149.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	17.0	27.0	47.0	98.0	4498.0
Differences	~	~	~	~	~	1600.0%	1250.0%	1075.0%	880.0%	291.47%

A.4.3 System level

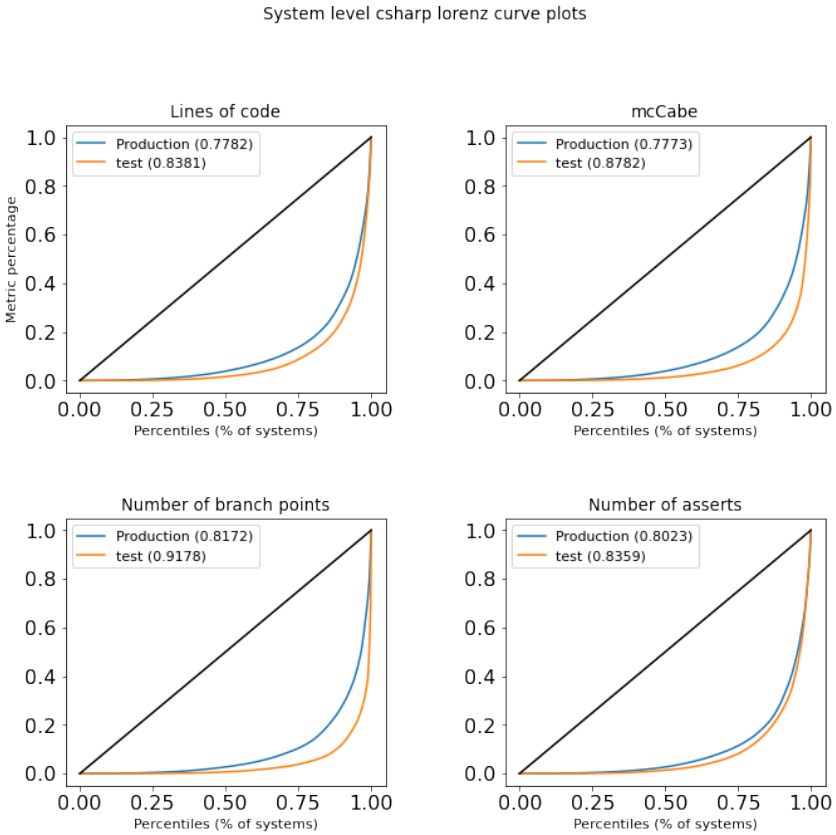

Figure A.23: Lorenz plots between CSharp production and CSharp test production systems.

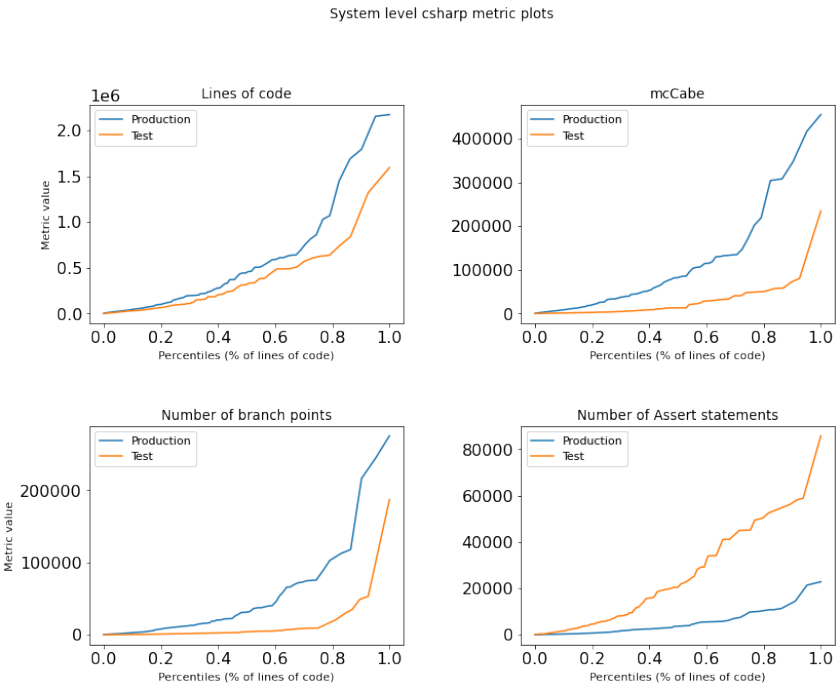

Figure A.24: Lorenz plots between CSharp production and CSharp test production systems.

Table A.43: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	44381.0	99098.0	194945.0	278093.0	442590.0	587560.0	691704.0	1069001.0	1689227.0	2154519.0
CSharp test SLOC	28175.0	64248.0	108472.0	204276.0	313855.0	437622.0	508226.0	636785.0	840121.0	1594573.0
Differences	57.52%	54.24%	79.72%	36.14%	41.02%	34.26%	36.1%	67.87%	101.07%	35.12%

Table A.44: Percentile values of CSharp production and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	8582.0	19168.0	37055.0	52958.0	81631.0	114470.0	133994.0	218716.0	307757.0	454793.0
CSharp test CC	1146.0	2592.0	4569.0	8364.0	12847.0	28037.0	40618.0	47865.0	71616.0	234164.0
Differences	648.87%	639.51%	711.01%	533.17%	535.41%	308.28%	229.89%	356.94%	329.73%	94.22%

Table A.45: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	2800.0	7958.0	12701.0	20300.0	30834.0	40252.0	72772.0	102699.0	117813.0	243970.0
CSharp test Branchpoints	256.0	760.0	1486.0	2359.0	3843.0	4963.0	8125.0	9188.0	48751.0	186983.0
Differences	993.75%	947.11%	754.71%	760.53%	702.34%	711.04%	795.66%	1017.75%	141.66%	30.48%

Table A.46: Percentile values of CSharp production and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	301.0	686.0	1676.0	2455.0	3567.0	5479.0	6907.0	10041.0	11276.0	21348.0
CSharp test asserts	1609.0	4538.0	8011.0	15738.0	20400.0	29171.0	41071.0	50276.0	56122.0	85742.0
Differences	434.55%	561.52%	377.98%	541.06%	471.91%	432.41%	494.63%	400.71%	397.71%	301.64%

Appendix B

Validation: Comparative analysis

B.1 Java production - CSharp production

B.1.1 Unit level

SLOC

Table B.1: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	35.0	70.0	1760.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	3689.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	42.86%	28.57%	109.6%

Table B.2: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	2062.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	3816.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	26.76%	85.06%

Table B.3: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	2062.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	91.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	28.17%	110.09%

Table B.4: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2739.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	28.57%	58.16%

Table B.5: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	24.0	36.0	72.0	2062.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	41.67%	38.89%	25.0%	110.09%

Table B.6: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	2062.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	26.76%	110.09%

Table B.7: Percentile values of Java production and CSharp production for the SLOC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	1603.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
Differences	0.0%	133.33%	66.67%	55.56%	50.0%	47.06%	47.83%	38.89%	28.57%	170.24%

Table B.8: Differences between Java production and CSharp production for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	133.33	66.67	55.56	50.0	47.06	47.83	42.86	28.57	109.60
1	0.0	133.33	66.67	55.56	50.0	47.06	47.83	38.89	26.76	85.06
2	0.0	133.33	66.67	55.56	50.0	47.06	47.83	38.89	28.17	110.09
3	0.0	133.33	66.67	55.56	50.0	47.06	47.83	38.89	28.57	58.16
4	0.0	133.33	66.67	55.56	50.0	47.06	41.67	38.89	25.00	110.09
5	0.0	133.33	66.67	55.56	50.0	47.06	47.83	38.89	26.76	110.09
6	0.0	133.33	66.67	55.56	50.0	47.06	47.83	38.89	28.57	170.24

Table B.9: Average differences between Java production and CSharp production for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	133.33	66.67	55.56	50.0	47.06	46.95	39.46	27.49	107.62

CC

Table B.10: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	81.27%

Table B.11: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	81.27%

Table B.12: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	562.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	6.41%

Table B.13: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	873.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	439.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	98.86%

Table B.14: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	562.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	38.08%

Table B.15: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	81.27%

Table B.16: Percentile values of Java production and CSharp production for the CC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	770.0
Differences	0.0%	0.0%	0.0%	0.0%	50.0%	33.33%	20.0%	28.57%	6.67%	28.76%

Table B.17: Differences between Java production and CSharp production for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	81.27
1	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	81.27
2	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	6.41
3	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	98.86
4	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	38.08
5	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	81.27
6	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	28.76

Table B.18: Average differences between Java production and CSharp production for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	0.0	0.0	0.0	50.0	33.33	20.0	28.57	6.67	59.42

Branchpoint

Table B.19: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	39.74%

Table B.20: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	39.74%

Table B.21: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	438.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	76.94%

Table B.22: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	468.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	443.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	5.64%

Table B.23: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	872.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	24.2%

Table B.24: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	872.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	438.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	99.09%

Table B.25: Percentile values of Java production and CSharp production for the Branchpoints metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	872.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	443.0
Differences	~	~	~	0.0%	100.0%	50.0%	25.0%	33.33%	7.14%	96.84%

Table B.26: Differences between Java production and CSharp production for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	39.74
1	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	39.74
2	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	76.94
3	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	5.64
4	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	24.20
5	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	99.09
6	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	96.84

Table B.27: Average differences between Java production and CSharp production for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	0.0	100.0	50.0	25.0	33.33	7.14	54.6

B.1.2 File level

SLOC

Table B.28: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	197.0	284.0	435.0	820.0	7238.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	245.0	387.0	665.0	1438.0	36664.0
Differences	26.09%	19.05%	7.35%	1.94%	11.97%	24.37%	36.27%	52.87%	75.37%	406.55%

Table B.29: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	102.0	140.0	195.0	280.0	426.0	787.0	9195.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	246.0	388.0	666.0	1436.0	20861.0
Differences	26.09%	19.05%	7.35%	2.94%	13.57%	26.15%	38.57%	56.34%	82.47%	126.87%

Table B.30: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	197.0	284.0	435.0	800.0	7551.0
CSharp production SLOC	23.0	43.0	69.0	107.0	161.0	250.0	395.0	679.0	1465.0	31690.0
Differences	26.09%	16.28%	5.8%	3.88%	13.38%	26.9%	39.08%	56.09%	83.12%	319.68%

Table B.31: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	196.0	283.0	436.0	813.0	24990.0
CSharp production SLOC	23.0	43.0	69.0	107.0	163.0	253.0	400.0	686.0	1483.0	31690.0
Differences	26.09%	16.28%	5.8%	3.88%	14.79%	29.08%	41.34%	57.34%	82.41%	26.81%

Table B.32: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	74.0	103.0	142.0	199.0	285.0	438.0	811.0	7551.0
CSharp production SLOC	23.0	42.0	69.0	106.0	161.0	248.0	395.0	674.0	1474.0	36664.0
Differences	26.09%	19.05%	7.25%	2.91%	13.38%	24.62%	38.6%	53.88%	81.75%	385.55%

Table B.33: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	102.0	141.0	196.0	283.0	433.0	806.0	7125.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	245.0	391.0	669.0	1436.0	31690.0
Differences	26.09%	19.05%	7.35%	2.94%	12.77%	25.0%	38.16%	54.5%	78.16%	344.77%

Table B.34: Percentile values of Java production and CSharp production for the SLOC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	141.0	197.0	283.0	435.0	798.0	9195.0
CSharp production SLOC	23.0	42.0	69.0	106.0	159.0	246.0	389.0	672.0	1440.0	28434.0
Differences	26.09%	19.05%	5.8%	2.91%	12.77%	24.87%	37.46%	54.48%	80.45%	209.23%

Table B.35: Differences between Java production and CSharp production for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	26.09	19.05	7.35	1.94	11.97	24.37	36.27	52.87	75.37	406.55
1	26.09	19.05	7.35	2.94	13.57	26.15	38.57	56.34	82.47	126.87
2	26.09	16.28	5.80	3.88	13.38	26.90	39.08	56.09	83.12	319.68
3	26.09	16.28	5.80	3.88	14.79	29.08	41.34	57.34	82.41	26.81
4	26.09	19.05	7.25	2.91	13.38	24.62	38.60	53.88	81.75	385.55
5	26.09	19.05	7.35	2.94	12.77	25.00	38.16	54.50	78.16	344.77
6	26.09	19.05	5.80	2.91	12.77	24.87	37.46	54.48	80.45	209.23

Table B.36: Average differences between Java production and CSharp production for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	26.09	18.26	6.67	3.06	13.23	25.86	38.5	55.07	80.53	259.92

CC

Table B.37: Percentile values of Java production and CSharp production for the CC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	55.0	87.0	172.0	2136.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	42.0	69.0	121.0	280.0	4221.0
Differences	0.0%	14.29%	18.18%	0.0%	3.85%	13.51%	25.45%	39.08%	62.79%	97.61%

Table B.38: Percentile values of Java production and CSharp production for the CC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	55.0	87.0	168.0	5539.0
CSharp production CC	4.0	7.0	11.0	17.0	27.0	42.0	67.0	116.0	253.0	2884.0
Differences	0.0%	14.29%	18.18%	5.88%	3.85%	13.51%	21.82%	33.33%	50.6%	92.06%

Table B.39: Percentile values of Java production and CSharp production for the CC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	85.0	166.0	2200.0
CSharp production CC	4.0	7.0	11.0	17.0	26.0	42.0	67.0	117.0	253.0	4872.0
Differences	0.0%	14.29%	18.18%	5.88%	0.0%	13.51%	24.07%	37.65%	52.41%	121.45%

Table B.40: Percentile values of Java production and CSharp production for the CC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	5.0	8.0	13.0	19.0	26.0	38.0	56.0	88.0	172.0	2136.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	69.0	121.0	267.0	3297.0
Differences	25.0%	14.29%	18.18%	5.56%	3.85%	13.16%	23.21%	37.5%	55.23%	54.35%

Table B.41: Percentile values of Java production and CSharp production for the CC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	55.0	88.0	174.0	2242.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	71.0	124.0	279.0	4221.0
Differences	0.0%	14.29%	18.18%	0.0%	3.85%	16.22%	29.09%	40.91%	60.34%	88.27%

Table B.42: Percentile values of Java production and CSharp production for the CC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	169.0	5539.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	42.0	69.0	120.0	269.0	3991.0
Differences	0.0%	14.29%	18.18%	0.0%	3.85%	13.51%	27.78%	39.53%	59.17%	38.79%

Table B.43: Percentile values of Java production and CSharp production for the CC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	55.0	86.0	168.0	3402.0
CSharp production CC	4.0	7.0	11.0	17.0	27.0	42.0	68.0	119.0	266.0	4872.0
Differences	0.0%	14.29%	18.18%	5.88%	3.85%	13.51%	23.64%	38.37%	58.33%	43.21%

Table B.44: Differences between Java production and CSharp production for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	14.29	18.18	0.00	3.85	13.51	25.45	39.08	62.79	97.61
1	0.0	14.29	18.18	5.88	3.85	13.51	21.82	33.33	50.60	92.06
2	0.0	14.29	18.18	5.88	0.00	13.51	24.07	37.65	52.41	121.45
3	25.0	14.29	18.18	5.56	3.85	13.16	23.21	37.50	55.23	54.35
4	0.0	14.29	18.18	0.00	3.85	16.22	29.09	40.91	60.34	88.27
5	0.0	14.29	18.18	0.00	3.85	13.51	27.78	39.53	59.17	38.79
6	0.0	14.29	18.18	5.88	3.85	13.51	23.64	38.37	58.33	43.21

Table B.45: Average differences between Java production and CSharp production for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	3.57	14.29	18.18	3.31	3.3	13.85	25.01	38.05	56.98	76.53

Branchpoints

Table B.46: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	15.0	26.0	48.0	103.0	1822.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	22.0	39.0	75.0	172.0	4349.0
Differences	~	~	0.0%	20.0%	33.33%	46.67%	50.0%	56.25%	66.99%	138.69%

Table B.47: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	48.0	104.0	1576.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	74.0	171.0	2298.0
Differences	~	~	0.0%	20.0%	33.33%	31.25%	44.44%	54.17%	64.42%	45.81%

Table B.48: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	103.0	1652.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	74.0	167.0	2016.0
Differences	~	~	0.0%	20.0%	33.33%	31.25%	44.44%	51.02%	62.14%	22.03%

Table B.49: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	104.0	1652.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	38.0	74.0	168.0	4349.0
Differences	~	~	0.0%	20.0%	33.33%	31.25%	40.74%	51.02%	61.54%	163.26%

Table B.50: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	103.0	1797.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	22.0	40.0	77.0	183.0	2974.0
Differences	~	~	0.0%	20.0%	33.33%	37.5%	48.15%	57.14%	77.67%	65.5%

Table B.51: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	106.0	1652.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	22.0	39.0	75.0	174.0	2974.0
Differences	~	~	0.0%	20.0%	33.33%	37.5%	44.44%	53.06%	64.15%	80.02%

Table B.52: Percentile values of Java production and CSharp production for the Branchpoints metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	104.0	2953.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	40.0	76.0	178.0	4349.0
Differences	~	~	0.0%	20.0%	33.33%	31.25%	48.15%	55.1%	71.15%	47.27%

Table B.53: Differences between Java production and CSharp production for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	0.0	20.0	33.33	46.67	50.00	56.25	66.99	138.69
1	NaN	NaN	0.0	20.0	33.33	31.25	44.44	54.17	64.42	45.81
2	NaN	NaN	0.0	20.0	33.33	31.25	44.44	51.02	62.14	22.03
3	NaN	NaN	0.0	20.0	33.33	31.25	40.74	51.02	61.54	163.26
4	NaN	NaN	0.0	20.0	33.33	37.50	48.15	57.14	77.67	65.50
5	NaN	NaN	0.0	20.0	33.33	37.50	44.44	53.06	64.15	80.02
6	NaN	NaN	0.0	20.0	33.33	31.25	48.15	55.10	71.15	47.27

Table B.54: Average differences between Java production and CSharp production for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	0.0	20.0	33.33	35.24	45.77	53.97	66.87	80.37

Asserts

Table B.55: Percentile values of Java production and CSharp production for the asserts metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	402.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	11.0	1149.0
Differences	~	~	~	~	~	~	~	~	~	185.82%

Table B.56: Percentile values of Java production and CSharp production for the asserts metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	714.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	3.0	10.0	572.0
Differences	~	~	~	~	~	~	~	~	~	24.83%

Table B.57: Percentile values of Java production and CSharp production for the asserts metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	11.0	1149.0
Differences	~	~	~	~	~	~	~	~	~	38.6%

Table B.58: Percentile values of Java production and CSharp production for the asserts metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	4.0	10.0	556.0
Differences	~	~	~	~	~	~	~	~	~	49.1%

Table B.59: Percentile values of Java production and CSharp production for the asserts metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1566.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	3.0	10.0	542.0
Differences	~	~	~	~	~	~	~	~	~	188.93%

Table B.60: Percentile values of Java production and CSharp production for the asserts metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1566.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	556.0
Differences	~	~	~	~	~	~	~	~	~	181.65%

Table B.61: Percentile values of Java production and CSharp production for the asserts metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1566.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	11.0	1149.0
Differences	~	~	~	~	~	~	~	~	~	36.29%

Table B.62: Differences between Java production and CSharp production for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	185.82
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.83
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	38.60
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	49.10
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	188.93
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	181.65
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	36.29

Table B.63: Average differences between Java production and CSharp production for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	100.75

Direct asserts

Table B.64: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.0	837.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	718.0
Differences	~	~	~	~	~	~	~	~	1000.0%	16.57%

Table B.65: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	845.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	718.0
Differences	~	~	~	~	~	~	~	~	900.0%	17.69%

Table B.66: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	838.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2766.0
Differences	~	~	~	~	~	~	~	~	900.0%	230.07%

Table B.67: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	1529.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	718.0
Differences	~	~	~	~	~	~	~	~	900.0%	112.95%

Table B.68: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	838.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1059.0
Differences	~	~	~	~	~	~	~	~	900.0%	26.37%

Table B.69: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	838.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2766.0
Differences	~	~	~	~	~	~	~	~	900.0%	230.07%

Table B.70: Percentile values of Java production and CSharp production for the Direct asserts metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11.0	845.0
CSharp production Direct asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1751.0
Differences	~	~	~	~	~	~	~	~	1000.0%	107.22%

Table B.71: Differences between Java production and CSharp production for the Direct asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1000.0	16.57
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	900.0	17.69
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	900.0	230.07
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	900.0	112.95
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	900.0	26.37
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	900.0	230.07
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1000.0	107.22

Table B.72: Average differences between Java production and CSharp production for the Direct asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	928.57	105.85

B.1.3 System level

SLOC

Table B.73: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	48530.0	90483.0	165955.0	222630.0	304662.0	554478.0	603134.0	942367.0	1200297.0	1462250.0
CSharp production SLOC	50089.0	116915.0	199273.0	278304.0	504964.0	639537.0	1029293.0	1447269.0	1795303.0	2173399.0
Differences	3.21%	29.21%	20.08%	25.01%	65.75%	15.34%	70.66%	53.58%	49.57%	48.63%

Table B.74: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	45078.0	92568.0	161981.0	248154.0	330314.0	563063.0	853485.0	1189979.0	1209227.0	1462250.0
CSharp production SLOC	54256.0	115467.0	235017.0	461164.0	587560.0	859273.0	1069001.0	1689227.0	1795303.0	2173399.0
Differences	20.36%	24.74%	45.09%	85.84%	77.88%	52.61%	25.25%	41.95%	48.47%	48.63%

Table B.75: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	46337.0	92674.0	163844.0	241729.0	330314.0	442009.0	668992.0	942367.0	1108624.0	1462250.0
CSharp production SLOC	44539.0	116915.0	199091.0	326973.0	504964.0	629363.0	816256.0	1069001.0	1447269.0	2173399.0
Differences	4.04%	26.16%	21.51%	35.26%	52.87%	42.39%	22.01%	13.44%	30.55%	48.63%

Table B.76: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	36868.0	75167.0	139723.0	212611.0	315657.0	442009.0	828862.0	853485.0	995766.0	1200297.0
CSharp production SLOC	48262.0	119002.0	194553.0	310173.0	504964.0	591154.0	859273.0	1069001.0	1795303.0	2154519.0
Differences	30.9%	58.32%	39.24%	45.89%	59.97%	33.74%	3.67%	25.25%	80.29%	79.5%

Table B.77: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	53493.0	112230.0	195851.0	304662.0	437712.0	572573.0	828862.0	1189979.0	1462250.0	1481603.0
CSharp production SLOC	54256.0	123483.0	217392.0	369512.0	461164.0	611108.0	691704.0	1029293.0	1447269.0	1795303.0
Differences	1.43%	10.03%	11.0%	21.29%	5.36%	6.73%	19.83%	15.61%	1.04%	21.17%

Table B.78: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	42901.0	92760.0	170010.0	246038.0	330314.0	668992.0	853485.0	995766.0	1189979.0	1481603.0
CSharp production SLOC	41028.0	98996.0	171138.0	235421.0	504964.0	610275.0	637991.0	816256.0	1689227.0	1689227.0
Differences	4.57%	6.72%	0.66%	4.51%	52.87%	9.62%	33.78%	21.99%	41.95%	14.01%

Table B.79: Percentile values of Java production and CSharp production for the SLOC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	43093.0	116701.0	199164.0	256137.0	423941.0	828862.0	942367.0	1200297.0	1462250.0	1481603.0
CSharp production SLOC	37864.0	86026.0	151184.0	217076.0	370296.0	535604.0	611108.0	816256.0	1795303.0	2154519.0
Differences	13.81%	35.66%	31.74%	17.99%	14.49%	54.75%	54.21%	47.05%	22.78%	45.42%

Table B.80: Differences between Java production and CSharp production for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	3.21	29.21	20.08	25.01	65.75	15.34	70.66	53.58	49.57	48.63
1	20.36	24.74	45.09	85.84	77.88	52.61	25.25	41.95	48.47	48.63
2	4.04	26.16	21.51	35.26	52.87	42.39	22.01	13.44	30.55	48.63
3	30.90	58.32	39.24	45.89	59.97	33.74	3.67	25.25	80.29	79.50
4	1.43	10.03	11.00	21.29	5.36	6.73	19.83	15.61	1.04	21.17
5	4.57	6.72	0.66	4.51	52.87	9.62	33.78	21.99	41.95	14.01
6	13.81	35.66	31.74	17.99	14.49	54.75	54.21	47.05	22.78	45.42

Table B.81: Average differences between Java production and CSharp production for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	11.19	27.26	24.19	33.68	47.03	30.74	32.77	31.27	39.24	43.71

CC

Table B.82: Percentile values of Java production and CSharp production for the CC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7899.0	16926.0	28142.0	37416.0	55967.0	76444.0	136336.0	195843.0	217534.0	218904.0
CSharp production CC	8575.0	17283.0	33388.0	50695.0	73594.0	85493.0	114470.0	201651.0	218716.0	416370.0
Differences	8.56%	2.11%	18.64%	35.49%	31.5%	11.84%	19.1%	2.97%	0.54%	90.21%

Table B.83: Percentile values of Java production and CSharp production for the CC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	6062.0	15324.0	24646.0	34991.0	50813.0	62108.0	104333.0	150545.0	180293.0	233887.0
CSharp production CC	8882.0	25065.0	34777.0	53346.0	81631.0	133038.0	201651.0	304054.0	307757.0	416370.0
Differences	46.52%	63.57%	41.11%	52.46%	60.65%	114.2%	93.28%	101.97%	70.7%	78.02%

Table B.84: Percentile values of Java production and CSharp production for the CC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	8017.0	18359.0	30802.0	40245.0	61038.0	77635.0	102332.0	197373.0	224999.0	275706.0
CSharp production CC	10238.0	25877.0	39938.0	52958.0	81631.0	106390.0	133994.0	169767.0	307757.0	416370.0
Differences	27.7%	40.95%	29.66%	31.59%	33.74%	37.04%	30.94%	16.26%	36.78%	51.02%

Table B.85: Percentile values of Java production and CSharp production for the CC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7328.0	15325.0	27592.0	37416.0	51709.0	66128.0	77939.0	171903.0	224999.0	233887.0
CSharp production CC	8202.0	20214.0	39179.0	52958.0	83365.0	114470.0	132217.0	201651.0	218716.0	416370.0
Differences	11.93%	31.9%	41.99%	41.54%	61.22%	73.1%	69.64%	17.31%	2.87%	78.02%

Table B.86: Percentile values of Java production and CSharp production for the CC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7474.0	18359.0	29886.0	51709.0	67040.0	136336.0	180293.0	217534.0	233887.0	332876.0
CSharp production CC	6466.0	15870.0	29067.0	44059.0	53346.0	83365.0	105532.0	133994.0	201651.0	416370.0
Differences	15.59%	15.68%	2.82%	17.36%	25.67%	63.54%	70.84%	62.35%	15.99%	25.08%

Table B.87: Percentile values of Java production and CSharp production for the CC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	9129.0	19918.0	47332.0	62108.0	76488.0	104333.0	180293.0	197373.0	233887.0	334705.0
CSharp production CC	11148.0	25591.0	43004.0	71250.0	118124.0	133038.0	169767.0	307757.0	346707.0	454793.0
Differences	22.12%	28.48%	10.06%	14.72%	54.43%	27.51%	6.2%	55.93%	48.24%	35.88%

Table B.88: Percentile values of Java production and CSharp production for the CC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7060.0	15955.0	29886.0	40251.0	61332.0	76488.0	104333.0	195843.0	224999.0	332876.0
CSharp production CC	8649.0	18101.0	33441.0	44059.0	62999.0	105532.0	133038.0	146752.0	346707.0	346707.0
Differences	22.51%	13.45%	11.9%	9.46%	2.72%	37.97%	27.51%	33.45%	54.09%	4.16%

Table B.89: Differences between Java production and CSharp production for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	8.56	2.11	18.64	35.49	31.50	11.84	19.10	2.97	0.54	90.21
1	46.52	63.57	41.11	52.46	60.65	114.20	93.28	101.97	70.70	78.02
2	27.70	40.95	29.66	31.59	33.74	37.04	30.94	16.26	36.78	51.02
3	11.93	31.90	41.99	41.54	61.22	73.10	69.64	17.31	2.87	78.02
4	15.59	15.68	2.82	17.36	25.67	63.54	70.84	62.35	15.99	25.08
5	22.12	28.48	10.06	14.72	54.43	27.51	6.20	55.93	48.24	35.88
6	22.51	13.45	11.90	9.46	2.72	37.97	27.51	33.45	54.09	4.16

Table B.90: Average differences between Java production and CSharp production for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	22.13	28.02	22.31	28.95	38.56	52.17	45.36	41.46	32.74	51.77

Branchpoints

Table B.91: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2356.0	5638.0	9741.0	13528.0	18515.0	28930.0	32979.0	54178.0	73326.0	187433.0
CSharp production Branchpoints	2291.0	5021.0	10245.0	12931.0	18856.0	31281.0	40252.0	74556.0	88928.0	216200.0
Differences	2.84%	12.29%	5.17%	4.62%	1.84%	8.13%	22.05%	37.61%	21.28%	15.35%

Table B.92: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2372.0	6052.0	11397.0	15531.0	20991.0	26548.0	54180.0	94075.0	139875.0	187433.0
CSharp production Branchpoints	2870.0	7068.0	12701.0	18583.0	31281.0	39314.0	66093.0	74556.0	112605.0	275225.0
Differences	20.99%	16.79%	11.44%	19.65%	49.02%	48.09%	21.99%	26.18%	24.22%	46.84%

Table B.93: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2990.0	6011.0	11397.0	18446.0	25758.0	28939.0	38970.0	73326.0	97642.0	139875.0
CSharp production Branchpoints	3074.0	8242.0	12868.0	22088.0	31281.0	66093.0	75497.0	102699.0	112605.0	243970.0
Differences	2.81%	37.12%	12.91%	19.74%	21.44%	128.39%	93.73%	40.06%	15.32%	74.42%

Table B.94: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	3570.0	7347.0	14675.0	24140.0	28939.0	73326.0	94847.0	97642.0	142658.0	187433.0
CSharp production Branchpoints	2291.0	5482.0	10615.0	14817.0	22825.0	36087.0	53986.0	72089.0	75497.0	112605.0
Differences	55.83%	34.02%	38.25%	62.92%	26.79%	103.19%	75.69%	35.45%	88.96%	66.45%

Table B.95: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2413.0	6268.0	10353.0	17949.0	25550.0	29472.0	35034.0	94075.0	97642.0	187433.0
CSharp production Branchpoints	3112.0	9210.0	13918.0	20687.0	36087.0	40252.0	74556.0	112605.0	117813.0	216200.0
Differences	28.97%	46.94%	34.43%	15.25%	41.24%	36.58%	112.81%	19.7%	20.66%	15.35%

Table B.96: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	1977.0	5762.0	8884.0	14675.0	21780.0	28939.0	35034.0	82654.0	97166.0	187433.0
CSharp production Branchpoints	2537.0	5183.0	10019.0	13918.0	20252.0	32052.0	53986.0	72089.0	75234.0	117813.0
Differences	28.33%	11.17%	12.78%	5.44%	7.54%	10.76%	54.1%	14.66%	29.15%	59.09%

Table B.97: Percentile values of Java production and CSharp production for the Branchpoints metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2933.0	7166.0	12582.0	25550.0	26683.0	32979.0	63057.0	94075.0	142658.0	187433.0
CSharp production Branchpoints	3074.0	7523.0	12701.0	21965.0	39314.0	58162.0	74556.0	88928.0	112605.0	275225.0
Differences	4.81%	4.98%	0.95%	16.32%	47.34%	76.36%	18.24%	5.79%	26.69%	46.84%

Table B.98: Differences between Java production and CSharp production for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	2.84	12.29	5.17	4.62	1.84	8.13	22.05	37.61	21.28	15.35
1	20.99	16.79	11.44	19.65	49.02	48.09	21.99	26.18	24.22	46.84
2	2.81	37.12	12.91	19.74	21.44	128.39	93.73	40.06	15.32	74.42
3	55.83	34.02	38.25	62.92	26.79	103.19	75.69	35.45	88.96	66.45
4	28.97	46.94	34.43	15.25	41.24	36.58	112.81	19.70	20.66	15.35
5	28.33	11.17	12.78	5.44	7.54	10.76	54.10	14.66	29.15	59.09
6	4.81	4.98	0.95	16.32	47.34	76.36	18.24	5.79	26.69	46.84

Table B.99: Average differences between Java production and CSharp production for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	20.65	23.33	16.56	20.56	27.89	58.79	56.94	25.64	32.33	46.33

Asserts

Table B.100: Percentile values of Java production and CSharp production for the asserts metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	2.0	7.0	39.0	89.0	175.0	309.0	1592.0
CSharp production asserts	281.0	664.0	1292.0	2445.0	3884.0	5555.0	7444.0	10771.0	14490.0	22815.0
Differences	~	~	~	122150.0%	55385.71%	14143.59%	8264.04%	6054.86%	4589.32%	1333.1%

Table B.101: Percentile values of Java production and CSharp production for the asserts metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	3.0	5.0	11.0	48.0	119.0	992.0	2459.0	25081.0
CSharp production asserts	260.0	685.0	1733.0	2372.0	2990.0	4945.0	5701.0	9748.0	21348.0	21348.0
Differences	~	~	57666.67%	47340.0%	27081.82%	10202.08%	4690.76%	882.66%	768.16%	17.49%

Table B.102: Percentile values of Java production and CSharp production for the asserts metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	2.0	3.0	10.0	46.0	119.0	492.0	24842.0
CSharp production asserts	317.0	730.0	1735.0	2445.0	3688.0	5587.0	6907.0	10041.0	10771.0	21348.0
Differences	~	~	~	122150.0%	122833.33%	55770.0%	14915.22%	8337.82%	2089.23%	16.37%

Table B.103: Percentile values of Java production and CSharp production for the asserts metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	4.0	12.0	37.0	116.0	421.0	992.0	2459.0
CSharp production asserts	299.0	685.0	1541.0	2191.0	3091.0	4515.0	7444.0	11276.0	14490.0	22815.0
Differences	~	~	76950.0%	54675.0%	25658.33%	12102.7%	6317.24%	2578.38%	1360.69%	827.82%

Table B.104: Percentile values of Java production and CSharp production for the asserts metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	3.0	5.0	12.0	61.0	119.0	2445.0	6948.0
CSharp production asserts	296.0	822.0	1734.0	2455.0	3688.0	5416.0	6907.0	10770.0	11033.0	22815.0
Differences	~	~	~	81733.33%	73660.0%	45033.33%	11222.95%	8950.42%	351.25%	228.37%

Table B.105: Percentile values of Java production and CSharp production for the asserts metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	5.0	21.0	68.0	175.0	992.0	6948.0	24842.0
CSharp production asserts	341.0	934.0	2139.0	2913.0	3932.0	5701.0	7444.0	11276.0	14490.0	22815.0
Differences	~	~	106850.0%	58160.0%	18623.81%	8283.82%	4153.71%	1036.69%	108.55%	8.88%

Table B.106: Percentile values of Java production and CSharp production for the asserts metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	3.0	3.0	5.0	13.0	46.0	421.0	24842.0
CSharp production asserts	231.0	377.0	917.0	1851.0	2372.0	3567.0	3932.0	5701.0	6092.0	10041.0
Differences	~	~	~	61600.0%	78966.67%	71240.0%	30146.15%	12293.48%	1347.03%	147.41%

Table B.107: Differences between Java production and CSharp production for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	122150.00	55385.71	14143.59	8264.04	6054.86	4589.32	1333.10
1	NaN	NaN	57666.67	47340.00	27081.82	10202.08	4690.76	882.66	768.16	17.49
2	NaN	NaN	NaN	122150.00	122833.33	55770.00	14915.22	8337.82	2089.23	16.37
3	NaN	NaN	76950.00	54675.00	25658.33	12102.70	6317.24	2578.38	1360.69	827.82
4	NaN	NaN	NaN	81733.33	73660.00	45033.33	11222.95	8950.42	351.25	228.37
5	NaN	NaN	106850.00	58160.00	18623.81	8283.82	4153.71	1036.69	108.55	8.88
6	NaN	NaN	NaN	61600.00	78966.67	71240.00	30146.15	12293.48	1347.03	147.41

Table B.108: Average differences between Java production and CSharp production for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	80488.89	78258.33	57458.52	30967.93	11387.15	5733.47	1516.32	368.49

Direct asserts

Table B.109: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	19.0	95.0	313.0	1134.0	1837.0	2549.0	4872.0	14080.0	35744.0	86309.0
CSharp production Direct asserts	22.0	178.0	660.0	1555.0	2154.0	2863.0	4003.0	8241.0	16522.0	38551.0
Differences	15.79%	87.37%	110.86%	37.13%	17.26%	12.32%	21.71%	70.85%	116.34%	123.88%

Table B.110: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	22.0	136.0	204.0	1038.0	2446.0	4492.0	13178.0	20304.0	35744.0	86309.0
CSharp production Direct asserts	41.0	202.0	801.0	1865.0	2766.0	3749.0	3830.0	8241.0	21436.0	52714.0
Differences	86.36%	48.53%	292.65%	79.67%	13.08%	19.82%	244.07%	146.38%	66.75%	63.73%

Table B.111: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	38.0	169.0	280.0	1294.0	1974.0	4490.0	12168.0	21609.0	35964.0	77048.0
CSharp production Direct asserts	3.0	103.0	394.0	885.0	996.0	2863.0	3751.0	12767.0	21436.0	52714.0
Differences	1166.67%	64.08%	40.71%	46.21%	98.19%	56.83%	224.39%	69.26%	67.77%	46.16%

Table B.112: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	13.0	169.0	373.0	1152.0	2333.0	6637.0	11945.0	31512.0	60947.0	86309.0
CSharp production Direct asserts	22.0	211.0	526.0	1002.0	1840.0	3802.0	4003.0	7469.0	15410.0	52714.0
Differences	69.23%	24.85%	41.02%	14.97%	26.79%	74.57%	198.4%	321.9%	295.5%	63.73%

Table B.113: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	98.0	167.0	280.0	1101.0	2024.0	3260.0	8437.0	15473.0	31512.0	77048.0
CSharp production Direct asserts	8.0	105.0	519.0	1794.0	2009.0	3082.0	3802.0	7469.0	12767.0	44589.0
Differences	1125.0%	59.05%	85.36%	62.94%	0.75%	5.78%	121.91%	107.16%	146.82%	72.8%

Table B.114: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	34.0	169.0	356.0	1152.0	2322.0	3260.0	6637.0	14364.0	35744.0	77048.0
CSharp production Direct asserts	7.0	296.0	821.0	885.0	1860.0	2017.0	3751.0	6669.0	11414.0	20761.0
Differences	385.71%	75.15%	130.62%	30.17%	24.84%	61.63%	76.94%	115.38%	213.16%	271.12%

Table B.115: Percentile values of Java production and CSharp production for the Direct asserts metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Direct asserts	55.0	204.0	364.0	1408.0	3323.0	6936.0	14080.0	22898.0	60947.0	86309.0
CSharp production Direct asserts	0.0	107.0	220.0	801.0	1520.0	2089.0	2381.0	7469.0	11414.0	44589.0
Differences	~	90.65%	65.45%	75.78%	118.62%	232.02%	491.35%	206.57%	433.97%	93.57%

Table B.116: Differences between Java production and CSharp production for the Direct asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	15.79	87.37	110.86	37.13	17.26	12.32	21.71	70.85	116.34	123.88
1	86.36	48.53	292.65	79.67	13.08	19.82	244.07	146.38	66.75	63.73
2	1166.67	64.08	40.71	46.21	98.19	56.83	224.39	69.26	67.77	46.16
3	69.23	24.85	41.02	14.97	26.79	74.57	198.40	321.90	295.50	63.73
4	1125.00	59.05	85.36	62.94	0.75	5.78	121.91	107.16	146.82	72.80
5	385.71	75.15	130.62	30.17	24.84	61.63	76.94	115.38	213.16	271.12
6	NaN	90.65	65.45	75.78	118.62	232.02	491.35	206.57	433.97	93.57

Table B.117: Average differences between Java production and CSharp production for the Direct asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	474.79	64.24	109.52	49.55	42.79	66.14	196.97	148.21	191.47	105.0

B.2 Java test - CSharp test

B.2.1 Unit level

SLOC

Table B.118: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	48.0	83.0	131.0	197.0	3605.0
Differences	75.0%	83.33%	100.0%	130.0%	166.67%	200.0%	315.0%	367.86%	319.15%	126.44%

Table B.119: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	22.0	32.0	47.0	79.0	127.0	194.0	3605.0
Differences	75.0%	83.33%	100.0%	120.0%	166.67%	193.75%	295.0%	353.57%	312.77%	126.44%

Table B.120: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	23.0	31.0	46.0	79.0	126.0	191.0	3605.0
Differences	75.0%	83.33%	100.0%	130.0%	158.33%	187.5%	295.0%	350.0%	306.38%	126.44%

Table B.121: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	48.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	22.0	31.0	46.0	78.0	125.0	185.0	3605.0
Differences	75.0%	83.33%	100.0%	120.0%	158.33%	187.5%	290.0%	346.43%	285.42%	126.44%

Table B.122: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
CSharp test SLOC	7.0	11.0	16.0	22.0	31.0	46.0	79.0	126.0	186.0	1661.0
Differences	75.0%	83.33%	100.0%	120.0%	158.33%	187.5%	295.0%	350.0%	295.74%	4.33%

Table B.123: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	962.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	48.0	82.0	130.0	195.0	15549.0
Differences	75.0%	83.33%	100.0%	130.0%	166.67%	200.0%	310.0%	364.29%	314.89%	1516.32%

Table B.124: Percentile values of Java test and CSharp test for the SLOC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	29.0	48.0	1001.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	47.0	81.0	128.0	190.0	1587.0
Differences	75.0%	83.33%	100.0%	130.0%	166.67%	193.75%	305.0%	341.38%	295.83%	58.54%

Table B.125: Differences between Java test and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	75.0	83.33	100.0	130.0	166.67	200.00	315.0	367.86	319.15	126.44
1	75.0	83.33	100.0	120.0	166.67	193.75	295.0	353.57	312.77	126.44
2	75.0	83.33	100.0	130.0	158.33	187.50	295.0	350.00	306.38	126.44
3	75.0	83.33	100.0	120.0	158.33	187.50	290.0	346.43	285.42	126.44
4	75.0	83.33	100.0	120.0	158.33	187.50	295.0	350.00	295.74	4.33
5	75.0	83.33	100.0	130.0	166.67	200.00	310.0	364.29	314.89	1516.32
6	75.0	83.33	100.0	130.0	166.67	193.75	305.0	341.38	295.83	58.54

Table B.126: Average differences between Java test and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	75.0	83.33	100.0	125.71	163.1	192.86	300.71	353.36	304.31	297.85

CC

Table B.127: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	71.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	331.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	366.2%

Table B.128: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	111.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	222.52%

Table B.129: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	15.0	25.0	331.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1400.0%	733.33%	171.31%

Table B.130: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	25.0	358.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	733.33%	193.44%

Table B.131: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	193.44%

Table B.132: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	106.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	331.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	212.26%

Table B.133: Percentile values of Java test and CSharp test for the CC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	276.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	400.0%	1500.0%	766.67%	126.23%

Table B.134: Differences between Java test and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	766.67	366.20
1	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	766.67	222.52
2	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1400.0	733.33	171.31
3	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	733.33	193.44
4	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	766.67	193.44
5	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	766.67	212.26
6	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1500.0	766.67	126.23

Table B.135: Average differences between Java test and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	0.0	0.0	0.0	0.0	100.0	400.0	1485.71	757.14	212.2

Branchpoints

Table B.136: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	110.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	~	~	~	1150.0%	224.55%

Table B.137: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	70.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	24.0	357.0
Differences	~	~	~	~	~	~	~	~	1100.0%	410.0%

Table B.138: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	80.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	275.0
Differences	~	~	~	~	~	~	~	~	1150.0%	243.75%

Table B.139: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	105.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	~	~	~	1150.0%	240.0%

Table B.140: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	110.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	269.0
Differences	~	~	~	~	~	~	~	~	1150.0%	144.55%

Table B.141: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	14.0	24.0	357.0
Differences	~	~	~	~	~	~	~	~	1100.0%	195.04%

Table B.142: Percentile values of Java test and CSharp test for the Branchpoints metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	~	~	~	1150.0%	195.04%

Table B.143: Differences between Java test and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1150.0	224.55
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1100.0	410.00
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1150.0	243.75
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1150.0	240.00
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1150.0	144.55
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1100.0	195.04
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1150.0	195.04

Table B.144: Average differences between Java test and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1135.71	236.13

B.2.2 File level

SLOC

Table B.145: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	178.0	240.0	331.0	484.0	854.0	13430.0
CSharp test SLOC	53.0	86.0	126.0	179.0	254.0	367.0	555.0	902.0	1804.0	19117.0
Differences	15.22%	21.13%	27.27%	34.59%	42.7%	52.92%	67.67%	86.36%	111.24%	42.35%

Table B.146: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	238.0	328.0	480.0	838.0	18933.0
CSharp test SLOC	52.0	85.0	125.0	176.0	249.0	362.0	549.0	872.0	1718.0	19117.0
Differences	13.04%	19.72%	26.26%	32.33%	40.68%	52.1%	67.38%	81.67%	105.01%	0.97%

Table B.147: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	45.0	71.0	99.0	132.0	175.0	236.0	325.0	474.0	818.0	13430.0
CSharp test SLOC	53.0	88.0	129.0	182.0	259.0	375.0	574.0	926.0	2019.0	68502.0
Differences	17.78%	23.94%	30.3%	37.88%	48.0%	58.9%	76.62%	95.36%	146.82%	410.07%

Table B.148: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	45.0	71.0	98.0	131.0	175.0	237.0	327.0	474.0	825.0	6728.0
CSharp test SLOC	53.0	86.0	126.0	179.0	253.0	367.0	563.0	917.0	1978.0	23666.0
Differences	17.78%	21.13%	28.57%	36.64%	44.57%	54.85%	72.17%	93.46%	139.76%	251.75%

Table B.149: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	240.0	331.0	481.0	836.0	18933.0
CSharp test SLOC	52.0	86.0	126.0	178.0	252.0	368.0	562.0	901.0	1754.0	28079.0
Differences	13.04%	21.13%	27.27%	33.83%	42.37%	53.33%	69.79%	87.32%	109.81%	48.31%

Table B.150: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	178.0	239.0	332.0	484.0	855.0	11505.0
CSharp test SLOC	53.0	87.0	128.0	181.0	258.0	377.0	581.0	949.0	1981.0	28079.0
Differences	15.22%	22.54%	29.29%	36.09%	44.94%	57.74%	75.0%	96.07%	131.7%	144.06%

Table B.151: Percentile values of Java test and CSharp test for the SLOC metric on a Files level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	238.0	328.0	481.0	851.0	11505.0
CSharp test SLOC	53.0	86.0	127.0	179.0	254.0	372.0	569.0	919.0	1932.0	68502.0
Differences	15.22%	21.13%	28.28%	34.59%	43.5%	56.3%	73.48%	91.06%	127.03%	495.41%

Table B.152: Differences between Java test and CSharp test for the SLOC metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	15.22	21.13	27.27	34.59	42.70	52.92	67.67	86.36	111.24	42.35
1	13.04	19.72	26.26	32.33	40.68	52.10	67.38	81.67	105.01	0.97
2	17.78	23.94	30.30	37.88	48.00	58.90	76.62	95.36	146.82	410.07
3	17.78	21.13	28.57	36.64	44.57	54.85	72.17	93.46	139.76	251.75
4	13.04	21.13	27.27	33.83	42.37	53.33	69.79	87.32	109.81	48.31
5	15.22	22.54	29.29	36.09	44.94	57.74	75.00	96.07	131.70	144.06
6	15.22	21.13	28.28	34.59	43.50	56.30	73.48	91.06	127.03	495.41

Table B.153: Average differences between Java test and CSharp test for the SLOC metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	15.33	21.53	28.18	35.14	43.82	55.16	71.73	90.19	124.48	198.99

CC

Table B.154: Percentile values of Java test and CSharp test for the CC metric on a Files level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	53.0	1906.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	43.0	127.0	2354.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	9.09%	23.53%	59.26%	139.62%	23.5%

Table B.155: Percentile values of Java test and CSharp test for the CC metric on a Files level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	28.0	54.0	4488.0
CSharp test CC	2.0	2.0	3.0	5.0	8.0	12.0	22.0	42.0	122.0	2868.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	9.09%	29.41%	50.0%	125.93%	56.49%

Table B.156: Percentile values of Java test and CSharp test for the CC metric on a Files level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	7.0	11.0	17.0	27.0	53.0	3368.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	42.0	121.0	2868.0
Differences	0.0%	0.0%	0.0%	0.0%	0.0%	9.09%	23.53%	55.56%	128.3%	17.43%

Table B.157: Percentile values of Java test and CSharp test for the CC metric on a Files level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	4488.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	20.0	40.0	118.0	2774.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	9.09%	17.65%	48.15%	126.92%	61.79%

Table B.158: Percentile values of Java test and CSharp test for the CC metric on a Files level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	7.0	11.0	17.0	27.0	52.0	908.0
CSharp test CC	2.0	2.0	3.0	5.0	8.0	13.0	22.0	44.0	133.0	2992.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	18.18%	29.41%	62.96%	155.77%	229.52%

Table B.159: Percentile values of Java test and CSharp test for the CC metric on a Files level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	53.0	1906.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	41.0	121.0	2972.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	9.09%	23.53%	51.85%	128.3%	55.93%

Table B.160: Percentile values of Java test and CSharp test for the CC metric on a Files level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	1626.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	43.0	121.0	2972.0
Differences	0.0%	0.0%	0.0%	0.0%	14.29%	9.09%	23.53%	59.26%	132.69%	82.78%

Table B.161: Differences between Java test and CSharp test for the CC metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	0.0	0.0	0.0	14.29	9.09	23.53	59.26	139.62	23.50
1	0.0	0.0	0.0	0.0	0.00	9.09	29.41	50.00	125.93	56.49
2	0.0	0.0	0.0	0.0	0.00	9.09	23.53	55.56	128.30	17.43
3	0.0	0.0	0.0	0.0	14.29	9.09	17.65	48.15	126.92	61.79
4	0.0	0.0	0.0	0.0	14.29	18.18	29.41	62.96	155.77	229.52
5	0.0	0.0	0.0	0.0	14.29	9.09	23.53	51.85	128.30	55.93
6	0.0	0.0	0.0	0.0	14.29	9.09	23.53	59.26	132.69	82.78

Table B.162: Average differences between Java test and CSharp test for the CC metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	0.0	0.0	0.0	10.21	10.39	24.37	55.29	133.93	75.35

Branchpoints

Table B.163: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	501.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	18.0	81.0	2443.0
Differences	~	~	~	~	~	0.0%	50.0%	100.0%	237.5%	387.62%

Table B.164: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	618.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	17.0	70.0	2454.0
Differences	~	~	~	~	~	0.0%	50.0%	88.89%	191.67%	297.09%

Table B.165: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	22.0	656.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	85.0	2709.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	286.36%	312.96%

Table B.166: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	501.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	84.0	2709.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	265.22%	440.72%

Table B.167: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	501.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	82.0	2828.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	256.52%	464.47%

Table B.168: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	618.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	18.0	77.0	2462.0
Differences	~	~	~	~	~	0.0%	50.0%	100.0%	220.83%	298.38%

Table B.169: Percentile values of Java test and CSharp test for the Branchpoints metric on a Files level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	501.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	80.0	2462.0
Differences	~	~	~	~	~	0.0%	50.0%	111.11%	233.33%	391.42%

Table B.170: Differences between Java test and CSharp test for the Branchpoints metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	0.0	50.0	100.00	237.50	387.62
1	NaN	NaN	NaN	NaN	NaN	0.0	50.0	88.89	191.67	297.09
2	NaN	NaN	NaN	NaN	NaN	0.0	50.0	111.11	286.36	312.96
3	NaN	NaN	NaN	NaN	NaN	0.0	50.0	111.11	265.22	440.72
4	NaN	NaN	NaN	NaN	NaN	0.0	50.0	111.11	256.52	464.47
5	NaN	NaN	NaN	NaN	NaN	0.0	50.0	100.00	220.83	298.38
6	NaN	NaN	NaN	NaN	NaN	0.0	50.0	111.11	233.33	391.42

Table B.171: Average differences between Java test and CSharp test for the Branchpoints metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	0.0	50.0	104.76	241.63	370.38

Asserts

Table B.172: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	1529.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	27.0	45.0	92.0	2227.0
Differences	~	0.0%	33.33%	16.67%	0.0%	0.0%	0.0%	0.0%	3.37%	45.65%

Table B.173: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	44.0	88.0	837.0
CSharp test asserts	0.0	1.0	4.0	7.0	11.0	18.0	27.0	47.0	101.0	2766.0
Differences	~	0.0%	0.0%	0.0%	0.0%	5.88%	0.0%	6.82%	14.77%	230.47%

Table B.174: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	88.0	961.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	26.0	45.0	93.0	1742.0
Differences	~	0.0%	33.33%	16.67%	0.0%	0.0%	3.85%	0.0%	5.68%	81.27%

Table B.175: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	28.0	46.0	91.0	883.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	27.0	48.0	104.0	4480.0
Differences	~	0.0%	33.33%	16.67%	0.0%	0.0%	3.7%	4.35%	14.29%	407.36%

Table B.176: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	961.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	27.0	45.0	96.0	4498.0
Differences	~	0.0%	33.33%	16.67%	0.0%	0.0%	0.0%	0.0%	7.87%	368.05%

Table B.177: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	89.0	913.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	17.0	27.0	48.0	103.0	4498.0
Differences	~	0.0%	33.33%	0.0%	0.0%	0.0%	0.0%	6.67%	15.73%	392.66%

Table B.178: Percentile values of Java test and CSharp test for the asserts metric on a Files level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	46.0	91.0	933.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	17.0	27.0	47.0	100.0	5430.0
Differences	~	0.0%	33.33%	0.0%	0.0%	0.0%	0.0%	2.17%	9.89%	481.99%

Table B.179: Differences between Java test and CSharp test for the asserts metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	0.0	33.33	16.67	0.0	0.00	0.00	0.00	3.37	45.65
1	NaN	0.0	0.00	0.00	0.0	5.88	0.00	6.82	14.77	230.47
2	NaN	0.0	33.33	16.67	0.0	0.00	3.85	0.00	5.68	81.27
3	NaN	0.0	33.33	16.67	0.0	0.00	3.70	4.35	14.29	407.36
4	NaN	0.0	33.33	16.67	0.0	0.00	0.00	0.00	7.87	368.05
5	NaN	0.0	33.33	0.00	0.0	0.00	0.00	6.67	15.73	392.66
6	NaN	0.0	33.33	0.00	0.0	0.00	0.00	2.17	9.89	481.99

Table B.180: Average differences between Java test and CSharp test for the asserts metric on a Files level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	0.0	28.57	9.53	0.0	0.84	1.08	2.86	10.23	286.78

B.2.3 System level

SLOC

Table B.181: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	17046.0	51230.0	87374.0	148694.0	190001.0	225068.0	283342.0	350883.0	470474.0	924809.0
CSharp test SLOC	21429.0	33671.0	54826.0	91134.0	156472.0	236928.0	283954.0	383936.0	487409.0	487409.0
Differences	25.71%	52.15%	59.37%	63.16%	21.43%	5.27%	0.22%	9.42%	3.6%	89.74%

Table B.182: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	19852.0	50077.0	113581.0	190001.0	306151.0	373606.0	544441.0	822316.0	843011.0	915395.0
CSharp test SLOC	27781.0	56057.0	88197.0	121224.0	215156.0	310596.0	487409.0	490118.0	840121.0	1319201.0
Differences	39.94%	11.94%	28.78%	56.74%	42.29%	20.29%	11.7%	67.78%	0.34%	44.11%

Table B.183: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	25870.0	71330.0	149495.0	225068.0	246584.0	339474.0	470474.0	822316.0	832327.0	843011.0
CSharp test SLOC	34631.0	84071.0	154955.0	236928.0	381047.0	508226.0	626708.0	840121.0	1319201.0	1594573.0
Differences	33.87%	17.86%	3.65%	5.27%	54.53%	49.71%	33.21%	2.17%	58.5%	89.15%

Table B.184: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	20616.0	43294.0	87374.0	127828.0	154072.0	225068.0	246584.0	373606.0	470474.0	844233.0
CSharp test SLOC	21558.0	49508.0	95582.0	183066.0	215156.0	313855.0	487409.0	606175.0	626708.0	840121.0
Differences	4.57%	14.35%	9.39%	43.21%	39.65%	39.45%	97.66%	62.25%	33.21%	0.49%

Table B.185: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	29366.0	71032.0	157101.0	219578.0	283342.0	460638.0	544441.0	822316.0	844233.0	924809.0
CSharp test SLOC	30774.0	64388.0	119767.0	208162.0	310596.0	333902.0	487409.0	490118.0	840121.0	1319201.0
Differences	4.79%	10.32%	31.17%	5.48%	9.62%	37.96%	11.7%	67.78%	0.49%	42.65%

Table B.186: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	19572.0	53530.0	87374.0	149495.0	225068.0	306151.0	373606.0	460638.0	843011.0	924809.0
CSharp test SLOC	28839.0	66616.0	108429.0	182619.0	236928.0	313855.0	383936.0	487409.0	606175.0	738055.0
Differences	47.35%	24.45%	24.1%	22.16%	5.27%	2.52%	2.76%	5.81%	39.07%	25.3%

Table B.187: Percentile values of Java test and CSharp test for the SLOC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test SLOC	27153.0	74458.0	122130.0	155106.0	219578.0	373606.0	470474.0	822316.0	832327.0	915395.0
CSharp test SLOC	20422.0	38757.0	59540.0	95820.0	141163.0	208162.0	250779.0	381047.0	437622.0	840121.0
Differences	32.96%	92.11%	105.12%	61.87%	55.55%	79.48%	87.61%	115.8%	90.19%	8.96%

Table B.188: Differences between Java test and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	25.71	52.15	59.37	63.16	21.43	5.27	0.22	9.42	3.60	89.74
1	39.94	11.94	28.78	56.74	42.29	20.29	11.70	67.78	0.34	44.11
2	33.87	17.86	3.65	5.27	54.53	49.71	33.21	2.17	58.50	89.15
3	4.57	14.35	9.39	43.21	39.65	39.45	97.66	62.25	33.21	0.49
4	4.79	10.32	31.17	5.48	9.62	37.96	11.70	67.78	0.49	42.65
5	47.35	24.45	24.10	22.16	5.27	2.52	2.76	5.81	39.07	25.30
6	32.96	92.11	105.12	61.87	55.55	79.48	87.61	115.80	90.19	8.96

Table B.189: Average differences between Java test and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	27.03	31.88	37.37	36.84	32.62	33.53	34.98	47.29	32.2	42.91

CC

Table B.190: Percentile values of Java test and CSharp test for the CC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	955.0	2262.0	4390.0	6953.0	8148.0	10234.0	12956.0	16859.0	29820.0	45021.0
CSharp test CC	1020.0	2166.0	4278.0	7950.0	20711.0	28037.0	31757.0	40618.0	58190.0	58190.0
Differences	6.81%	4.43%	2.62%	14.34%	154.19%	173.96%	145.11%	140.93%	95.14%	29.25%

Table B.191: Percentile values of Java test and CSharp test for the CC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1183.0	3300.0	6953.0	10183.0	12956.0	21862.0	47795.0	66771.0	81549.0	104533.0
CSharp test CC	1004.0	2118.0	3154.0	5826.0	8364.0	12807.0	12924.0	28037.0	32836.0	71616.0
Differences	17.83%	55.81%	120.45%	74.79%	54.9%	70.7%	269.82%	138.15%	148.35%	45.96%

Table B.192: Percentile values of Java test and CSharp test for the CC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1097.0	2060.0	5861.0	8972.0	12592.0	13015.0	21862.0	36566.0	47309.0	66771.0
CSharp test CC	704.0	1831.0	2592.0	3753.0	6801.0	12034.0	28037.0	29675.0	32836.0	71616.0
Differences	55.82%	12.51%	126.12%	139.06%	85.15%	8.15%	28.25%	23.22%	44.08%	7.26%

Table B.193: Percentile values of Java test and CSharp test for the CC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	799.0	1954.0	3305.0	5861.0	8748.0	10234.0	12806.0	14084.0	21690.0	87887.0
CSharp test CC	1060.0	2118.0	4278.0	8944.0	12924.0	28037.0	40618.0	40618.0	50091.0	58190.0
Differences	32.67%	8.39%	29.44%	52.6%	47.74%	173.96%	217.18%	188.4%	130.94%	51.03%

Table B.194: Percentile values of Java test and CSharp test for the CC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	939.0	2749.0	4390.0	8017.0	12592.0	13015.0	16859.0	36566.0	45021.0	66771.0
CSharp test CC	1146.0	2574.0	5069.0	8746.0	12794.0	12924.0	28037.0	32836.0	58190.0	80217.0
Differences	22.04%	6.8%	15.47%	9.09%	1.6%	0.7%	66.3%	11.36%	29.25%	20.14%

Table B.195: Percentile values of Java test and CSharp test for the CC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	1121.0	3088.0	6798.0	8999.0	13602.0	22495.0	29820.0	47309.0	66771.0	104533.0
CSharp test CC	1020.0	2008.0	3591.0	8314.0	10999.0	12847.0	31757.0	40618.0	57269.0	71616.0
Differences	9.9%	53.78%	89.31%	8.24%	23.67%	75.1%	6.5%	16.47%	16.59%	45.96%

Table B.196: Percentile values of Java test and CSharp test for the CC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test CC	616.0	1452.0	2262.0	3099.0	5768.0	6953.0	9397.0	12806.0	16779.0	66771.0
CSharp test CC	1054.0	2166.0	3659.0	5923.0	10374.0	12924.0	28037.0	32836.0	71616.0	234164.0
Differences	71.1%	49.17%	61.76%	91.13%	79.85%	85.88%	198.36%	156.41%	326.82%	250.7%

Table B.197: Differences between Java test and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	6.81	4.43	2.62	14.34	154.19	173.96	145.11	140.93	95.14	29.25
1	17.83	55.81	120.45	74.79	54.90	70.70	269.82	138.15	148.35	45.96
2	55.82	12.51	126.12	139.06	85.15	8.15	28.25	23.22	44.08	7.26
3	32.67	8.39	29.44	52.60	47.74	173.96	217.18	188.40	130.94	51.03
4	22.04	6.80	15.47	9.09	1.60	0.70	66.30	11.36	29.25	20.14
5	9.90	53.78	89.31	8.24	23.67	75.10	6.50	16.47	16.59	45.96
6	71.10	49.17	61.76	91.13	79.85	85.88	198.36	156.41	326.82	250.70

Table B.198: Average differences between Java test and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	30.88	27.27	63.6	55.61	63.87	84.06	133.07	96.42	113.02	64.33

Branchpoints

Table B.199: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	283.0	581.0	893.0	1600.0	2465.0	3722.0	4401.0	7256.0	11207.0	33025.0
CSharp test Branchpoints	289.0	968.0	2259.0	2759.0	4763.0	5508.0	8974.0	9188.0	53057.0	186983.0
Differences	2.12%	66.61%	152.97%	72.44%	93.23%	47.98%	103.91%	26.63%	373.43%	466.19%

Table B.200: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	165.0	413.0	660.0	1299.0	1818.0	2100.0	3458.0	5271.0	10829.0	18823.0
CSharp test Branchpoints	411.0	1244.0	2298.0	2759.0	4494.0	6509.0	9188.0	34603.0	53057.0	186983.0
Differences	149.09%	201.21%	248.18%	112.39%	147.19%	209.95%	165.7%	556.48%	389.95%	893.38%

Table B.201: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	235.0	660.0	1356.0	1584.0	2134.0	3458.0	3707.0	7256.0	11207.0	33025.0
CSharp test Branchpoints	285.0	632.0	1278.0	2266.0	4494.0	7598.0	9188.0	20622.0	31078.0	186983.0
Differences	21.28%	4.43%	6.1%	43.06%	110.59%	119.72%	147.86%	184.21%	177.31%	466.19%

Table B.202: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	272.0	557.0	990.0	1818.0	3707.0	3722.0	5487.0	8826.0	11185.0	18823.0
CSharp test Branchpoints	236.0	439.0	929.0	1486.0	2298.0	3843.0	4763.0	6509.0	9128.0	9188.0
Differences	15.25%	26.88%	6.57%	22.34%	61.31%	3.25%	15.2%	35.6%	22.54%	104.87%

Table B.203: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	195.0	473.0	730.0	1356.0	1818.0	2391.0	2465.0	5271.0	7256.0	12527.0
CSharp test Branchpoints	268.0	789.0	1486.0	2259.0	2617.0	4494.0	5596.0	8125.0	31078.0	53057.0
Differences	37.44%	66.81%	103.56%	66.59%	43.95%	87.95%	127.02%	54.15%	328.31%	323.54%

Table B.204: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	245.0	637.0	1388.0	3164.0	4841.0	6321.0	8826.0	12527.0	33025.0	34157.0
CSharp test Branchpoints	280.0	789.0	1604.0	2407.0	4494.0	7598.0	8974.0	9188.0	20622.0	48751.0
Differences	14.29%	23.86%	15.56%	31.45%	7.72%	20.2%	1.68%	36.34%	60.14%	42.73%

Table B.205: Percentile values of Java test and CSharp test for the Branchpoints metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test Branchpoints	211.0	581.0	1299.0	1824.0	3707.0	4401.0	5487.0	7256.0	10829.0	33025.0
CSharp test Branchpoints	289.0	789.0	1912.0	2298.0	3843.0	4963.0	8125.0	9128.0	9188.0	31078.0
Differences	36.97%	35.8%	47.19%	25.99%	3.67%	12.77%	48.08%	25.8%	17.86%	6.26%

Table B.206: Differences between Java test and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	2.12	66.61	152.97	72.44	93.23	47.98	103.91	26.63	373.43	466.19
1	149.09	201.21	248.18	112.39	147.19	209.95	165.70	556.48	389.95	893.38
2	21.28	4.43	6.10	43.06	110.59	119.72	147.86	184.21	177.31	466.19
3	15.25	26.88	6.57	22.34	61.31	3.25	15.20	35.60	22.54	104.87
4	37.44	66.81	103.56	66.59	43.95	87.95	127.02	54.15	328.31	323.54
5	14.29	23.86	15.56	31.45	7.72	20.20	1.68	36.34	60.14	42.73
6	36.97	35.80	47.19	25.99	3.67	12.77	48.08	25.80	17.86	6.26

Table B.207: Average differences between Java test and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	39.49	60.8	82.88	53.47	66.81	71.69	87.06	131.32	195.65	329.02

Asserts

Table B.208: Percentile values of Java test and CSharp test for the asserts metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1600.0	4071.0	7950.0	14063.0	24411.0	39744.0	49274.0	90070.0	92921.0	96968.0
CSharp test asserts	1544.0	3910.0	5754.0	9398.0	16725.0	41071.0	41071.0	44926.0	49336.0	58844.0
Differences	3.63%	4.12%	38.16%	49.64%	45.96%	3.34%	19.97%	100.49%	88.34%	64.79%

Table B.209: Percentile values of Java test and CSharp test for the asserts metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1600.0	4382.0	7545.0	10084.0	17628.0	26800.0	36823.0	39744.0	90070.0	103755.0
CSharp test asserts	1417.0	3910.0	6477.0	14135.0	21865.0	25297.0	29171.0	41071.0	49336.0	58844.0
Differences	12.91%	12.07%	16.49%	40.17%	24.04%	5.94%	26.23%	3.34%	82.56%	76.32%

Table B.210: Percentile values of Java test and CSharp test for the asserts metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1156.0	3774.0	5608.0	9230.0	14856.0	19858.0	23731.0	36768.0	49274.0	92921.0
CSharp test asserts	2286.0	5698.0	11654.0	18674.0	29171.0	41071.0	45135.0	52477.0	58053.0	85742.0
Differences	97.75%	50.98%	107.81%	102.32%	96.36%	106.82%	90.19%	42.72%	17.82%	8.37%

Table B.211: Percentile values of Java test and CSharp test for the asserts metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	2099.0	5804.0	12410.0	21127.0	30006.0	49274.0	66636.0	90070.0	92921.0	103755.0
CSharp test asserts	1362.0	3910.0	8627.0	16725.0	20400.0	22725.0	33885.0	50276.0	58053.0	85742.0
Differences	54.11%	48.44%	43.85%	26.32%	47.09%	116.83%	96.65%	79.15%	60.06%	21.01%

Table B.212: Percentile values of Java test and CSharp test for the asserts metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	1851.0	4815.0	7950.0	12410.0	19858.0	26800.0	41968.0	53607.0	92921.0	96968.0
CSharp test asserts	1609.0	4606.0	8196.0	16725.0	22725.0	41071.0	44926.0	49336.0	58844.0	85742.0
Differences	15.04%	4.54%	3.09%	34.77%	14.44%	53.25%	7.05%	8.66%	57.91%	13.09%

Table B.213: Percentile values of Java test and CSharp test for the asserts metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	2241.0	5074.0	8838.0	17834.0	26585.0	39744.0	53607.0	66636.0	90070.0	103755.0
CSharp test asserts	1548.0	3834.0	7143.0	11654.0	18674.0	20400.0	49336.0	50276.0	58844.0	58844.0
Differences	44.77%	32.34%	23.73%	53.03%	42.36%	94.82%	8.66%	32.54%	53.07%	76.32%

Table B.214: Percentile values of Java test and CSharp test for the asserts metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java test asserts	2021.0	5466.0	8913.0	12691.0	20576.0	24680.0	36823.0	41968.0	53607.0	90070.0
CSharp test asserts	2286.0	5115.0	8011.0	12653.0	20400.0	28192.0	34168.0	45135.0	50276.0	56122.0
Differences	13.11%	6.86%	11.26%	0.3%	0.86%	14.23%	7.77%	7.55%	6.63%	60.49%

Table B.215: Differences between Java test and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	3.63	4.12	38.16	49.64	45.96	3.34	19.97	100.49	88.34	64.79
1	12.91	12.07	16.49	40.17	24.04	5.94	26.23	3.34	82.56	76.32
2	97.75	50.98	107.81	102.32	96.36	106.82	90.19	42.72	17.82	8.37
3	54.11	48.44	43.85	26.32	47.09	116.83	96.65	79.15	60.06	21.01
4	15.04	4.54	3.09	34.77	14.44	53.25	7.05	8.66	57.91	13.09
5	44.77	32.34	23.73	53.03	42.36	94.82	8.66	32.54	53.07	76.32
6	13.11	6.86	11.26	0.30	0.86	14.23	7.77	7.55	6.63	60.49

Table B.216: Average differences between Java test and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	34.47	22.76	34.91	43.79	38.73	56.46	36.65	39.21	52.34	45.77

B.3 Java production - Java test

B.3.1 Unit level

SLOC

Table B.217: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2062.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1001.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	48.94%	105.99%

Table B.218: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	1760.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1001.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	51.06%	75.82%

Table B.219: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2238.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	46.0	962.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	52.17%	132.64%

Table B.220: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2238.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	29.0	48.0	1012.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	24.14%	45.83%	121.15%

Table B.221: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	2062.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	15.0	20.0	28.0	46.0	1001.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	13.33%	15.0%	28.57%	54.35%	105.99%

Table B.222: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	71.0	2238.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	47.0	1592.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	51.06%	40.58%

Table B.223: Percentile values of Java production and Java test for the SLOC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	3.0	3.0	6.0	9.0	12.0	17.0	23.0	36.0	70.0	2739.0
Java test SLOC	4.0	6.0	8.0	10.0	12.0	16.0	20.0	28.0	48.0	1012.0
Differences	33.33%	100.0%	33.33%	11.11%	0.0%	6.25%	15.0%	28.57%	45.83%	170.65%

Table B.224: Differences between Java production and Java test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	33.33	100.0	33.33	11.11	0.0	6.25	15.0	28.57	48.94	105.99
1	33.33	100.0	33.33	11.11	0.0	6.25	15.0	28.57	51.06	75.82
2	33.33	100.0	33.33	11.11	0.0	6.25	15.0	28.57	52.17	132.64
3	33.33	100.0	33.33	11.11	0.0	6.25	15.0	24.14	45.83	121.15
4	33.33	100.0	33.33	11.11	0.0	13.33	15.0	28.57	54.35	105.99
5	33.33	100.0	33.33	11.11	0.0	6.25	15.0	28.57	51.06	40.58
6	33.33	100.0	33.33	11.11	0.0	6.25	15.0	28.57	45.83	170.65

Table B.225: Average differences between Java production and Java test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	33.33	100.0	33.33	11.11	0.0	7.26	15.0	27.94	49.89	107.55

CC

Table B.226: Percentile values of Java production and Java test for the CC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	111.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	599.1%

Table B.227: Percentile values of Java production and Java test for the CC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	776.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	75.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	934.67%

Table B.228: Percentile values of Java production and Java test for the CC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	390.16%

Table B.229: Percentile values of Java production and Java test for the CC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	873.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	106.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	723.58%

Table B.230: Percentile values of Java production and Java test for the CC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	598.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	81.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	638.27%

Table B.231: Percentile values of Java production and Java test for the CC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	873.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	122.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	615.57%

Table B.232: Percentile values of Java production and Java test for the CC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	1.0	1.0	1.0	2.0	2.0	3.0	5.0	7.0	15.0	873.0
Java test CC	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	3.0	111.0
Differences	0.0%	0.0%	0.0%	100.0%	100.0%	200.0%	400.0%	600.0%	400.0%	686.49%

Table B.233: Differences between Java production and Java test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	599.10
1	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	934.67
2	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	390.16
3	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	723.58
4	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	638.27
5	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	615.57
6	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	686.49

Table B.234: Average differences between Java production and Java test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	0.0	0.0	100.0	100.0	200.0	400.0	600.0	400.0	655.41

Branchpoints

Table B.235: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	570.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	105.0
Differences	~	~	~	~	~	~	~	~	600.0%	442.86%

Table B.236: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	375.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
Differences	~	~	~	~	~	~	~	~	600.0%	209.92%

Table B.237: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	872.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	79.0
Differences	~	~	~	~	~	~	~	~	600.0%	1003.8%

Table B.238: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	872.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	121.0
Differences	~	~	~	~	~	~	~	~	600.0%	620.66%

Table B.239: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	706.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	79.0
Differences	~	~	~	~	~	~	~	~	600.0%	793.67%

Table B.240: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	706.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	105.0
Differences	~	~	~	~	~	~	~	~	600.0%	572.38%

Table B.241: Percentile values of Java production and Java test for the Branchpoints metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	0.0	1.0	1.0	2.0	4.0	6.0	14.0	775.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	110.0
Differences	~	~	~	~	~	~	~	~	600.0%	604.55%

Table B.242: Differences between Java production and Java test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	442.86
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	209.92
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	1003.80
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	620.66
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	793.67
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	572.38
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	604.55

Table B.243: Average differences between Java production and Java test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	600.0	606.83

B.3.2 File level

SLOC

Table B.244: Percentile values of Java production and Java test for the SLOC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	141.0	196.0	281.0	429.0	794.0	7238.0
Java test SLOC	45.0	71.0	99.0	132.0	176.0	238.0	327.0	476.0	836.0	18933.0
Differences	55.17%	42.0%	35.62%	28.16%	24.82%	21.43%	16.37%	10.96%	5.29%	161.58%

Table B.245: Percentile values of Java production and Java test for the SLOC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	103.0	142.0	198.0	286.0	436.0	804.0	7125.0
Java test SLOC	46.0	71.0	99.0	132.0	176.0	236.0	328.0	478.0	850.0	11505.0
Differences	58.62%	42.0%	35.62%	28.16%	23.94%	19.19%	14.69%	9.63%	5.72%	61.47%

Table B.246: Percentile values of Java production and Java test for the SLOC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	74.0	103.0	143.0	199.0	286.0	441.0	817.0	24990.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	240.0	332.0	481.0	836.0	11505.0
Differences	58.62%	42.0%	33.78%	29.13%	23.78%	20.6%	16.08%	9.07%	2.33%	117.21%

Table B.247: Percentile values of Java production and Java test for the SLOC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	102.0	141.0	196.0	281.0	434.0	805.0	9195.0
Java test SLOC	46.0	71.0	99.0	133.0	178.0	242.0	335.0	497.0	899.0	18933.0
Differences	58.62%	42.0%	35.62%	30.39%	26.24%	23.47%	19.22%	14.52%	11.68%	105.91%

Table B.248: Percentile values of Java production and Java test for the SLOC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	74.0	103.0	143.0	199.0	288.0	446.0	824.0	9195.0
Java test SLOC	45.0	71.0	99.0	132.0	176.0	235.0	325.0	476.0	824.0	11505.0
Differences	55.17%	42.0%	33.78%	28.16%	23.08%	18.09%	12.85%	6.73%	0.0%	25.12%

Table B.249: Percentile values of Java production and Java test for the SLOC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	74.0	103.0	143.0	199.0	286.0	440.0	821.0	7125.0
Java test SLOC	46.0	71.0	99.0	133.0	177.0	239.0	330.0	485.0	849.0	18933.0
Differences	58.62%	42.0%	33.78%	29.13%	23.78%	20.1%	15.38%	10.23%	3.41%	165.73%

Table B.250: Percentile values of Java production and Java test for the SLOC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	29.0	50.0	73.0	102.0	141.0	196.0	280.0	428.0	786.0	7238.0
Java test SLOC	45.0	71.0	98.0	131.0	175.0	234.0	319.0	472.0	811.0	6728.0
Differences	55.17%	42.0%	34.25%	28.43%	24.11%	19.39%	13.93%	10.28%	3.18%	7.58%

Table B.251: Differences between Java production and Java test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	55.17	42.0	35.62	28.16	24.82	21.43	16.37	10.96	5.29	161.58
1	58.62	42.0	35.62	28.16	23.94	19.19	14.69	9.63	5.72	61.47
2	58.62	42.0	33.78	29.13	23.78	20.60	16.08	9.07	2.33	117.21
3	58.62	42.0	35.62	30.39	26.24	23.47	19.22	14.52	11.68	105.91
4	55.17	42.0	33.78	28.16	23.08	18.09	12.85	6.73	0.00	25.12
5	58.62	42.0	33.78	29.13	23.78	20.10	15.38	10.23	3.41	165.73
6	55.17	42.0	34.25	28.43	24.11	19.39	13.93	10.28	3.18	7.58

Table B.252: Average differences between Java production and Java test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	57.14	42.0	34.64	28.79	24.25	20.32	15.5	10.2	4.52	92.09

CC

Table B.253: Percentile values of Java production and Java test for the CC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	169.0	2136.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	53.0	4488.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	218.52%	218.87%	110.11%

Table B.254: Percentile values of Java production and Java test for the CC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	167.0	2136.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	3368.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	218.52%	221.15%	57.68%

Table B.255: Percentile values of Java production and Java test for the CC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	171.0	3402.0
Java test CC	2.0	2.0	3.0	5.0	7.0	11.0	17.0	27.0	52.0	1626.0
Differences	100.0%	300.0%	333.33%	260.0%	271.43%	236.36%	217.65%	218.52%	228.85%	109.23%

Table B.256: Percentile values of Java production and Java test for the CC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	5.0	8.0	13.0	18.0	26.0	37.0	54.0	86.0	168.0	2240.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	53.0	908.0
Differences	150.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	218.52%	216.98%	146.7%

Table B.257: Percentile values of Java production and Java test for the CC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	85.0	165.0	2240.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	28.0	54.0	1446.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	203.57%	205.56%	54.91%

Table B.258: Percentile values of Java production and Java test for the CC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	54.0	87.0	172.0	3402.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	53.0	4488.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	217.65%	222.22%	224.53%	31.92%

Table B.259: Percentile values of Java production and Java test for the CC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	4.0	8.0	13.0	18.0	26.0	37.0	55.0	86.0	171.0	5539.0
Java test CC	2.0	2.0	3.0	5.0	8.0	11.0	17.0	27.0	52.0	3368.0
Differences	100.0%	300.0%	333.33%	260.0%	225.0%	236.36%	223.53%	218.52%	228.85%	64.46%

Table B.260: Differences between Java production and Java test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	100.0	300.0	333.33	260.0	225.00	236.36	217.65	218.52	218.87	110.11
1	100.0	300.0	333.33	260.0	225.00	236.36	217.65	218.52	221.15	57.68
2	100.0	300.0	333.33	260.0	271.43	236.36	217.65	218.52	228.85	109.23
3	150.0	300.0	333.33	260.0	225.00	236.36	217.65	218.52	216.98	146.70
4	100.0	300.0	333.33	260.0	225.00	236.36	217.65	203.57	205.56	54.91
5	100.0	300.0	333.33	260.0	225.00	236.36	217.65	222.22	224.53	31.92
6	100.0	300.0	333.33	260.0	225.00	236.36	223.53	218.52	228.85	64.46

Table B.261: Average differences between Java production and Java test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	107.14	300.0	333.33	260.0	231.63	236.36	218.49	216.91	220.68	82.14

Branchpoints

Table B.262: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	48.0	104.0	2953.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	22.0	618.0
Differences	~	~	~	~	~	700.0%	575.0%	433.33%	372.73%	377.83%

Table B.263: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	106.0	2953.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	656.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	341.67%	350.15%

Table B.264: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	107.0	1963.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	488.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	345.83%	302.25%

Table B.265: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	15.0	27.0	48.0	103.0	1576.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	22.0	656.0
Differences	~	~	~	~	~	650.0%	575.0%	433.33%	368.18%	140.24%

Table B.266: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	105.0	1652.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	488.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	337.5%	238.52%

Table B.267: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	105.0	1822.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	23.0	392.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	356.52%	364.8%

Table B.268: Percentile values of Java production and Java test for the Branchpoints metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	0.0	0.0	2.0	5.0	9.0	16.0	27.0	49.0	102.0	1652.0
Java test Branchpoints	0.0	0.0	0.0	0.0	0.0	2.0	4.0	9.0	24.0	488.0
Differences	~	~	~	~	~	700.0%	575.0%	444.44%	325.0%	238.52%

Table B.269: Differences between Java production and Java test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	700.0	575.0	433.33	372.73	377.83
1	NaN	NaN	NaN	NaN	NaN	700.0	575.0	444.44	341.67	350.15
2	NaN	NaN	NaN	NaN	NaN	700.0	575.0	444.44	345.83	302.25
3	NaN	NaN	NaN	NaN	NaN	650.0	575.0	433.33	368.18	140.24
4	NaN	NaN	NaN	NaN	NaN	700.0	575.0	444.44	337.50	238.52
5	NaN	NaN	NaN	NaN	NaN	700.0	575.0	444.44	356.52	364.80
6	NaN	NaN	NaN	NaN	NaN	700.0	575.0	444.44	325.00	238.52

Table B.270: Average differences between Java production and Java test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	692.86	575.0	441.27	349.63	287.47

Asserts

Table B.271: Percentile values of Java production and Java test for the asserts metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	714.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	87.0	883.0
Differences	~	~	~	~	~	~	~	~	~	23.67%

Table B.272: Percentile values of Java production and Java test for the asserts metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1566.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	90.0	933.0
Differences	~	~	~	~	~	~	~	~	~	67.85%

Table B.273: Percentile values of Java production and Java test for the asserts metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	714.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	28.0	46.0	90.0	1529.0
Differences	~	~	~	~	~	~	~	~	~	114.15%

Table B.274: Percentile values of Java production and Java test for the asserts metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1566.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	18.0	28.0	47.0	91.0	961.0
Differences	~	~	~	~	~	~	~	~	~	62.96%

Table B.275: Percentile values of Java production and Java test for the asserts metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	90.0	961.0
Differences	~	~	~	~	~	~	~	~	~	15.92%

Table B.276: Percentile values of Java production and Java test for the asserts metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	714.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	45.0	88.0	1529.0
Differences	~	~	~	~	~	~	~	~	~	114.15%

Table B.277: Percentile values of Java production and Java test for the asserts metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	829.0
Java test asserts	0.0	1.0	4.0	7.0	11.0	18.0	28.0	46.0	89.0	961.0
Differences	~	~	~	~	~	~	~	~	~	15.92%

Table B.278: Differences between Java production and Java test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	23.67
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	67.85
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	114.15
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	62.96
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	15.92
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	114.15
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	15.92

Table B.279: Average differences between Java production and Java test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	59.23

B.3.3 System level

SLOC

Table B.280: Percentile values of Java production and Java test for the SLOC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	47839.0	115557.0	167567.0	309931.0	554478.0	668992.0	839063.0	942367.0	1200297.0	1481603.0
Java test SLOC	15514.0	38995.0	79875.0	135591.0	171621.0	243947.0	246584.0	373606.0	544441.0	924809.0
Differences	208.36%	196.34%	109.79%	128.58%	223.08%	174.24%	240.27%	152.24%	120.46%	60.21%

Table B.281: Percentile values of Java production and Java test for the SLOC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	57360.0	144952.0	233402.0	309931.0	558695.0	828862.0	995766.0	1189979.0	1462250.0	1481603.0
Java test SLOC	21878.0	52035.0	85757.0	135603.0	190001.0	246056.0	306151.0	460638.0	544441.0	822316.0
Differences	162.18%	178.57%	172.17%	128.56%	194.05%	236.86%	225.25%	158.33%	168.58%	80.17%

Table B.282: Percentile values of Java production and Java test for the SLOC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	42752.0	92568.0	187277.0	249696.0	330314.0	558695.0	668992.0	942367.0	1200297.0	1481603.0
Java test SLOC	30294.0	71032.0	149495.0	190001.0	254543.0	544441.0	832327.0	843011.0	844233.0	924809.0
Differences	41.12%	30.32%	25.27%	31.42%	29.77%	2.62%	24.42%	11.79%	42.18%	60.21%

Table B.283: Percentile values of Java production and Java test for the SLOC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	50856.0	112230.0	195851.0	248154.0	437712.0	668992.0	942367.0	1108624.0	1200297.0	1481603.0
Java test SLOC	32593.0	83906.0	155106.0	246056.0	339474.0	460638.0	544441.0	832327.0	843011.0	924809.0
Differences	56.03%	33.76%	26.27%	0.85%	28.94%	45.23%	73.09%	33.2%	42.38%	60.21%

Table B.284: Percentile values of Java production and Java test for the SLOC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	42901.0	86084.0	129307.0	187277.0	248154.0	423941.0	761771.0	906480.0	1189979.0	1481603.0
Java test SLOC	29366.0	62244.0	117138.0	155106.0	225068.0	339474.0	373606.0	470474.0	843011.0	915395.0
Differences	46.09%	38.3%	10.39%	20.74%	10.26%	24.88%	103.9%	92.67%	41.16%	61.85%

Table B.285: Percentile values of Java production and Java test for the SLOC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	43014.0	92674.0	153206.0	246038.0	335453.0	554478.0	828862.0	1108624.0	1209227.0	1481603.0
Java test SLOC	28882.0	71330.0	127828.0	184947.0	283342.0	460638.0	544441.0	822316.0	844233.0	924809.0
Differences	48.93%	29.92%	19.85%	33.03%	18.39%	20.37%	52.24%	34.82%	43.23%	60.21%

Table B.286: Percentile values of Java production and Java test for the SLOC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production SLOC	37737.0	76428.0	134430.0	202745.0	249696.0	437712.0	603134.0	853485.0	1108624.0	1200297.0
Java test SLOC	22523.0	53980.0	95892.0	149495.0	219578.0	283342.0	339474.0	470474.0	843011.0	844233.0
Differences	67.55%	41.59%	40.19%	35.62%	13.72%	54.48%	77.67%	81.41%	31.51%	42.18%

Table B.287: Differences between Java production and Java test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	208.36	196.34	109.79	128.58	223.08	174.24	240.27	152.24	120.46	60.21
1	162.18	178.57	172.17	128.56	194.05	236.86	225.25	158.33	168.58	80.17
2	41.12	30.32	25.27	31.42	29.77	2.62	24.42	11.79	42.18	60.21
3	56.03	33.76	26.27	0.85	28.94	45.23	73.09	33.20	42.38	60.21
4	46.09	38.30	10.39	20.74	10.26	24.88	103.90	92.67	41.16	61.85
5	48.93	29.92	19.85	33.03	18.39	20.37	52.24	34.82	43.23	60.21
6	67.55	41.59	40.19	35.62	13.72	54.48	77.67	81.41	31.51	42.18

Table B.288: Average differences between Java production and Java test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	90.04	78.4	57.7	54.11	74.03	79.81	113.83	80.64	69.93	60.72

CC

Table B.289: Percentile values of Java production and Java test for the CC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	9300.0	18672.0	29846.0	50813.0	66128.0	90449.0	150545.0	217534.0	224999.0	332876.0
Java test CC	1009.0	2262.0	4390.0	6953.0	8748.0	10656.0	16779.0	36566.0	47795.0	47795.0
Differences	821.7%	725.46%	579.86%	630.81%	655.92%	748.81%	797.22%	494.91%	370.76%	596.47%

Table B.290: Percentile values of Java production and Java test for the CC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	12138.0	27697.0	50813.0	63211.0	77939.0	150545.0	197373.0	218904.0	275706.0	332876.0
Java test CC	1097.0	2147.0	3930.0	8017.0	12592.0	16779.0	36566.0	45021.0	87887.0	104533.0
Differences	1006.47%	1190.03%	1192.95%	688.46%	518.96%	797.22%	439.77%	386.23%	213.71%	218.44%

Table B.291: Percentile values of Java production and Java test for the CC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7131.0	16926.0	29486.0	40245.0	61038.0	77900.0	195843.0	217534.0	218904.0	332876.0
Java test CC	958.0	2731.0	5768.0	9397.0	15173.0	21862.0	36566.0	45021.0	47795.0	81549.0
Differences	644.36%	519.77%	411.2%	328.27%	302.28%	256.33%	435.59%	383.18%	358.01%	308.19%

Table B.292: Percentile values of Java production and Java test for the CC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	10246.0	21295.0	38110.0	62108.0	77635.0	136336.0	197373.0	217534.0	275706.0	332876.0
Java test CC	1094.0	2894.0	4315.0	7727.0	10656.0	13602.0	36566.0	47309.0	47795.0	87887.0
Differences	836.56%	635.83%	783.2%	703.78%	628.56%	902.32%	439.77%	359.82%	476.85%	278.75%

Table B.293: Percentile values of Java production and Java test for the CC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	6050.0	15355.0	26843.0	38110.0	60119.0	77635.0	104333.0	180293.0	217534.0	233887.0
Java test CC	1094.0	2462.0	4390.0	8999.0	12301.0	16859.0	26453.0	45021.0	66771.0	87887.0
Differences	453.02%	523.68%	511.46%	323.49%	388.73%	360.5%	294.41%	300.46%	225.79%	166.12%

Table B.294: Percentile values of Java production and Java test for the CC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	7022.0	18199.0	35235.0	67040.0	101122.0	171903.0	197373.0	233887.0	275706.0	332876.0
Java test CC	1305.0	3986.0	8148.0	10656.0	13015.0	16859.0	29820.0	45021.0	47309.0	104533.0
Differences	438.08%	356.57%	332.44%	529.13%	676.97%	919.65%	561.88%	419.51%	482.78%	218.44%

Table B.295: Percentile values of Java production and Java test for the CC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production CC	6492.0	14446.0	27592.0	35235.0	51658.0	67040.0	101506.0	150545.0	224999.0	332876.0
Java test CC	952.0	1926.0	4298.0	6403.0	9323.0	13602.0	22495.0	29820.0	47309.0	104533.0
Differences	581.93%	650.05%	541.97%	450.29%	454.09%	392.87%	351.24%	404.85%	375.59%	218.44%

Table B.296: Differences between Java production and Java test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	821.70	725.46	579.86	630.81	655.92	748.81	797.22	494.91	370.76	596.47
1	1006.47	1190.03	1192.95	688.46	518.96	797.22	439.77	386.23	213.71	218.44
2	644.36	519.77	411.20	328.27	302.28	256.33	435.59	383.18	358.01	308.19
3	836.56	635.83	783.20	703.78	628.56	902.32	439.77	359.82	476.85	278.75
4	453.02	523.68	511.46	323.49	388.73	360.50	294.41	300.46	225.79	166.12
5	438.08	356.57	332.44	529.13	676.97	919.65	561.88	419.51	482.78	218.44
6	581.93	650.05	541.97	450.29	454.09	392.87	351.24	404.85	375.59	218.44

Table B.297: Average differences between Java production and Java test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	683.16	657.34	621.87	522.03	517.93	625.39	474.27	392.71	357.64	286.41

Branchpoints

Table B.298: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2411.0	5717.0	9741.0	15903.0	25550.0	35034.0	54178.0	94075.0	142658.0	187433.0
Java test Branchpoints	262.0	676.0	1388.0	1818.0	3164.0	4401.0	5487.0	8881.0	18823.0	34532.0
Differences	820.23%	745.71%	601.8%	774.75%	707.52%	696.05%	887.39%	959.28%	657.89%	442.78%

Table B.299: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	1987.0	5762.0	10353.0	15531.0	22035.0	31578.0	54178.0	63057.0	82654.0	159284.0
Java test Branchpoints	215.0	537.0	1133.0	1485.0	1818.0	2100.0	3722.0	3722.0	5932.0	11185.0
Differences	824.19%	973.0%	813.77%	945.86%	1112.05%	1403.71%	1355.62%	1594.17%	1293.36%	1324.09%

Table B.300: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	3030.0	6516.0	12387.0	17463.0	25758.0	31850.0	73326.0	94847.0	97642.0	187433.0
Java test Branchpoints	246.0	557.0	1036.0	1584.0	2134.0	3707.0	4841.0	7256.0	8826.0	18823.0
Differences	1131.71%	1069.84%	1095.66%	1002.46%	1107.03%	759.19%	1414.69%	1207.15%	1006.3%	895.77%

Table B.301: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	3323.0	7247.0	12021.0	25782.0	29472.0	54178.0	82654.0	97166.0	98948.0	187433.0
Java test Branchpoints	326.0	819.0	1356.0	1818.0	3458.0	4401.0	5932.0	8826.0	11185.0	34532.0
Differences	919.33%	784.86%	786.5%	1318.15%	752.28%	1131.04%	1293.36%	1000.91%	784.65%	442.78%

Table B.302: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2632.0	6955.0	15186.0	23017.0	27169.0	48609.0	82654.0	97166.0	98948.0	187433.0
Java test Branchpoints	221.0	487.0	741.0	1375.0	2111.0	3458.0	3722.0	7256.0	8826.0	34157.0
Differences	1090.95%	1328.13%	1949.39%	1573.96%	1187.02%	1305.7%	2120.69%	1239.11%	1021.1%	448.74%

Table B.303: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	3042.0	6268.0	12387.0	17275.0	25758.0	32979.0	54178.0	63057.0	142658.0	187433.0
Java test Branchpoints	211.0	493.0	843.0	1353.0	1780.0	2100.0	4841.0	10829.0	11207.0	34532.0
Differences	1341.71%	1171.4%	1369.4%	1176.79%	1347.08%	1470.43%	1019.15%	482.3%	1172.94%	442.78%

Table B.304: Percentile values of Java production and Java test for the Branchpoints metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production Branchpoints	2414.0	6386.0	12021.0	18061.0	29472.0	35034.0	73326.0	97642.0	142658.0	187433.0
Java test Branchpoints	246.0	730.0	1353.0	1987.0	2673.0	5347.0	7256.0	8826.0	8881.0	34532.0
Differences	881.3%	774.79%	788.47%	808.96%	1002.58%	555.21%	910.56%	1006.3%	1506.33%	442.78%

Table B.305: Differences between Java production and Java test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	820.23	745.71	601.80	774.75	707.52	696.05	887.39	959.28	657.89	442.78
1	824.19	973.00	813.77	945.86	1112.05	1403.71	1355.62	1594.17	1293.36	1324.09
2	1131.71	1069.84	1095.66	1002.46	1107.03	759.19	1414.69	1207.15	1006.30	895.77
3	919.33	784.86	786.50	1318.15	752.28	1131.04	1293.36	1000.91	784.65	442.78
4	1090.95	1328.13	1949.39	1573.96	1187.02	1305.70	2120.69	1239.11	1021.10	448.74
5	1341.71	1171.40	1369.40	1176.79	1347.08	1470.43	1019.15	482.30	1172.94	442.78
6	881.30	774.79	788.47	808.96	1002.58	555.21	910.56	1006.30	1506.33	442.78

Table B.306: Average differences between Java production and Java test for the Branch-points metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	1001.35	978.25	1057.86	1085.85	1030.79	1045.9	1285.92	1069.89	1063.22	634.25

Asserts

Table B.307: Percentile values of Java production and Java test for the asserts metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	5.0	13.0	56.0	358.0	992.0	6948.0
Java test asserts	987.0	2643.0	5466.0	9230.0	14492.0	18737.0	23316.0	26800.0	49274.0	49274.0
Differences	~	~	273200.0%	307566.67%	289740.0%	144030.77%	41535.71%	7386.03%	4867.14%	609.18%

Table B.308: Percentile values of Java production and Java test for the asserts metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	4.0	10.0	30.0	88.0	992.0	2359.0	25081.0
Java test asserts	2361.0	6956.0	14856.0	21127.0	36823.0	49274.0	66636.0	90070.0	92921.0	103755.0
Differences	~	~	742700.0%	528075.0%	368130.0%	164146.67%	75622.73%	8979.64%	3839.0%	313.68%

Table B.309: Percentile values of Java production and Java test for the asserts metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	3.0	5.0	27.0	100.0	175.0	469.0	1264.0	24842.0
Java test asserts	2518.0	5230.0	13852.0	20576.0	36768.0	49274.0	66636.0	92921.0	96129.0	103755.0
Differences	~	~	461633.33%	411420.0%	136077.78%	49174.0%	37977.71%	19712.58%	7505.14%	317.66%

Table B.310: Percentile values of Java production and Java test for the asserts metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	3.0	4.0	9.0	26.0	119.0	358.0	992.0	24842.0
Java test asserts	1796.0	4470.0	8632.0	17628.0	21127.0	26800.0	41968.0	53607.0	66636.0	103755.0
Differences	~	~	287633.33%	440600.0%	234644.44%	102976.92%	35167.23%	14874.02%	6617.34%	317.66%

Table B.311: Percentile values of Java production and Java test for the asserts metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	4.0	9.0	37.0	267.0	492.0	2459.0	24842.0
Java test asserts	2361.0	5644.0	9022.0	14856.0	19858.0	49274.0	66636.0	90070.0	92921.0	96129.0
Differences	~	~	451000.0%	371300.0%	220544.44%	133072.97%	24857.3%	18206.91%	3678.81%	286.96%

Table B.312: Percentile values of Java production and Java test for the asserts metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	0.0	2.0	3.0	9.0	12.0	49.0	358.0	2459.0
Java test asserts	883.0	2467.0	4781.0	8838.0	11358.0	14492.0	19858.0	26585.0	90070.0	90070.0
Differences	~	~	~	441800.0%	378500.0%	160922.22%	165383.33%	54155.1%	25059.22%	3562.87%

Table B.313: Percentile values of Java production and Java test for the asserts metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Java production asserts	0.0	0.0	2.0	3.0	7.0	14.0	89.0	492.0	1592.0	25081.0
Java test asserts	2038.0	5173.0	8838.0	12828.0	18670.0	23731.0	36823.0	49274.0	90070.0	103755.0
Differences	-	-	441800.0%	427500.0%	266614.29%	169407.14%	41274.16%	9915.04%	5557.66%	313.68%

Table B.314: Differences between Java production and Java test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	273200.00	307566.67	289740.00	144030.77	41535.71	7386.03	4867.14	609.18
1	NaN	NaN	742700.00	528075.00	368130.00	164146.67	75622.73	8979.64	3839.00	313.68
2	NaN	NaN	461633.33	411420.00	136077.78	49174.00	37977.71	19712.58	7505.14	317.66
3	NaN	NaN	287633.33	440600.00	234644.44	102976.92	35167.23	14874.02	6617.34	317.66
4	NaN	NaN	451000.00	371300.00	220544.44	133072.97	24857.30	18206.91	3678.81	286.96
5	NaN	NaN	NaN	441800.00	378500.00	160922.22	165383.33	54155.10	25059.22	3562.87
6	NaN	NaN	441800.00	427500.00	266614.29	169407.14	41274.16	9915.04	5557.66	313.68

Table B.315: Average differences between Java production and Java test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	442994.44	418323.1	270607.28	131961.53	60259.74	19032.76	8160.62	817.38

B.4 CSharp production - CSharp test

B.4.1 Unit level

SLOC

Table B.316: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	47.0	80.0	128.0	193.0	1511.0
Differences	133.33%	57.14%	60.0%	64.29%	77.78%	88.0%	135.29%	156.0%	114.44%	186.7%

Table B.317: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	91.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	47.0	80.0	127.0	191.0	15549.0
Differences	133.33%	57.14%	60.0%	64.29%	77.78%	88.0%	135.29%	154.0%	109.89%	258.93%

Table B.318: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	22.0	31.0	46.0	79.0	127.0	187.0	15549.0
Differences	133.33%	57.14%	60.0%	57.14%	72.22%	84.0%	132.35%	154.0%	107.78%	258.93%

Table B.319: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	91.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	47.0	80.0	128.0	189.0	1587.0
Differences	133.33%	57.14%	60.0%	64.29%	77.78%	88.0%	135.29%	156.0%	107.69%	172.97%

Table B.320: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	89.0	3816.0
CSharp test SLOC	7.0	11.0	16.0	22.0	31.0	46.0	79.0	126.0	186.0	1661.0
Differences	133.33%	57.14%	60.0%	57.14%	72.22%	84.0%	132.35%	152.0%	108.99%	129.74%

Table B.321: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	89.0	3689.0
CSharp test SLOC	7.0	11.0	16.0	22.0	31.0	46.0	78.0	126.0	186.0	1587.0
Differences	133.33%	57.14%	60.0%	57.14%	72.22%	84.0%	129.41%	152.0%	108.99%	132.45%

Table B.322: Percentile values of CSharp production and CSharp test for the SLOC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	3.0	7.0	10.0	14.0	18.0	25.0	34.0	50.0	90.0	4332.0
CSharp test SLOC	7.0	11.0	16.0	23.0	32.0	47.0	80.0	127.0	190.0	3605.0
Differences	133.33%	57.14%	60.0%	64.29%	77.78%	88.0%	135.29%	154.0%	111.11%	20.17%

Table B.323: Differences between CSharp production and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	133.33	57.14	60.0	64.29	77.78	88.0	135.29	156.0	114.44	186.70
1	133.33	57.14	60.0	64.29	77.78	88.0	135.29	154.0	109.89	258.93
2	133.33	57.14	60.0	57.14	72.22	84.0	132.35	154.0	107.78	258.93
3	133.33	57.14	60.0	64.29	77.78	88.0	135.29	156.0	107.69	172.97
4	133.33	57.14	60.0	57.14	72.22	84.0	132.35	152.0	108.99	129.74
5	133.33	57.14	60.0	57.14	72.22	84.0	129.41	152.0	108.99	132.45
6	133.33	57.14	60.0	64.29	77.78	88.0	135.29	154.0	111.11	20.17

Table B.324: Average differences between CSharp production and CSharp test for the SLOC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	133.33	57.14	60.0	61.23	75.4	86.29	133.61	154.0	109.84	165.7

CC

Table B.325: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	331.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	227.49%

Table B.326: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	304.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	256.58%

Table B.327: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	15.0	26.0	331.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	66.67%	62.5%	227.49%

Table B.328: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	331.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	227.49%

Table B.329: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	25.0	331.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	56.25%	227.49%

Table B.330: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	770.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	304.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	153.29%

Table B.331: Percentile values of CSharp production and CSharp test for the CC metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	1.0	1.0	1.0	2.0	3.0	4.0	6.0	9.0	16.0	1084.0
CSharp test CC	1.0	1.0	1.0	1.0	1.0	2.0	5.0	16.0	26.0	358.0
Differences	0.0%	0.0%	0.0%	100.0%	200.0%	100.0%	20.0%	77.78%	62.5%	202.79%

Table B.332: Differences between CSharp production and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	62.50	227.49
1	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	62.50	256.58
2	0.0	0.0	0.0	100.0	200.0	100.0	20.0	66.67	62.50	227.49
3	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	62.50	227.49
4	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	56.25	227.49
5	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	62.50	153.29
6	0.0	0.0	0.0	100.0	200.0	100.0	20.0	77.78	62.50	202.79

Table B.333: Average differences between CSharp production and CSharp test for the CC metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	0.0	0.0	0.0	100.0	200.0	100.0	20.0	76.19	61.61	217.52

Branchpoints

Table B.334: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	443.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	14.0	24.0	275.0
Differences	~	~	~	~	~	200.0%	25.0%	75.0%	60.0%	61.09%

Table B.335: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	5.0	15.0	25.0	303.0
Differences	~	~	~	~	~	200.0%	0.0%	87.5%	66.67%	257.43%

Table B.336: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	769.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	24.0	293.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	60.0%	162.46%

Table B.337: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	24.0	357.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	60.0%	203.36%

Table B.338: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	279.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	66.67%	288.17%

Table B.339: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	1083.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	279.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	66.67%	288.17%

Table B.340: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a Unit level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	0.0	1.0	2.0	3.0	5.0	8.0	15.0	438.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	0.0	1.0	4.0	15.0	25.0	357.0
Differences	~	~	~	~	~	200.0%	25.0%	87.5%	66.67%	22.69%

Table B.341: Differences between CSharp production and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	200.0	25.0	75.0	60.00	61.09
1	NaN	NaN	NaN	NaN	NaN	200.0	0.0	87.5	66.67	257.43
2	NaN	NaN	NaN	NaN	NaN	200.0	25.0	87.5	60.00	162.46
3	NaN	NaN	NaN	NaN	NaN	200.0	25.0	87.5	60.00	203.36
4	NaN	NaN	NaN	NaN	NaN	200.0	25.0	87.5	66.67	288.17
5	NaN	NaN	NaN	NaN	NaN	200.0	25.0	87.5	66.67	288.17
6	NaN	NaN	NaN	NaN	NaN	200.0	25.0	87.5	66.67	22.69

Table B.342: Average differences between CSharp production and CSharp test for the Branchpoints metric on a Unit level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	200.0	21.43	85.71	63.81	183.34

B.4.2 File level

SLOC

Table B.343: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	69.0	106.0	159.0	246.0	390.0	667.0	1440.0	36664.0
CSharp test SLOC	53.0	86.0	125.0	177.0	249.0	362.0	543.0	877.0	1845.0	28079.0
Differences	130.43%	104.76%	81.16%	66.98%	56.6%	47.15%	39.23%	31.48%	28.12%	30.57%

Table B.344: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	69.0	106.0	160.0	247.0	392.0	674.0	1437.0	36664.0
CSharp test SLOC	52.0	86.0	125.0	177.0	250.0	363.0	546.0	888.0	1782.0	23314.0
Differences	126.09%	104.76%	81.16%	66.98%	56.25%	46.96%	39.29%	31.75%	24.01%	57.26%

Table B.345: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	69.0	106.0	160.0	248.0	391.0	672.0	1432.0	36664.0
CSharp test SLOC	53.0	87.0	127.0	180.0	255.0	372.0	569.0	929.0	1918.0	68502.0
Differences	130.43%	107.14%	84.06%	69.81%	59.38%	50.0%	45.52%	38.24%	33.94%	86.84%

Table B.346: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	68.0	105.0	158.0	244.0	382.0	654.0	1346.0	31690.0
CSharp test SLOC	53.0	86.0	126.0	178.0	253.0	366.0	552.0	878.0	1803.0	68502.0
Differences	130.43%	104.76%	85.29%	69.52%	60.13%	50.0%	44.5%	34.25%	33.95%	116.16%

Table B.347: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	245.0	387.0	661.0	1410.0	28434.0
CSharp test SLOC	53.0	86.0	126.0	179.0	254.0	367.0	566.0	892.0	1855.0	23666.0
Differences	130.43%	104.76%	85.29%	70.48%	59.75%	49.8%	46.25%	34.95%	31.56%	20.15%

Table B.348: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	245.0	385.0	654.0	1411.0	31690.0
CSharp test SLOC	53.0	86.0	126.0	178.0	250.0	363.0	552.0	885.0	1769.0	23666.0
Differences	130.43%	104.76%	85.29%	69.52%	57.23%	48.16%	43.38%	35.32%	25.37%	33.91%

Table B.349: Percentile values of CSharp production and CSharp test for the SLOC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	23.0	42.0	68.0	105.0	159.0	244.0	384.0	656.0	1368.0	20861.0
CSharp test SLOC	53.0	86.0	126.0	179.0	254.0	370.0	567.0	912.0	1845.0	23666.0
Differences	130.43%	104.76%	85.29%	70.48%	59.75%	51.64%	47.66%	39.02%	34.87%	13.45%

Table B.350: Differences between CSharp production and CSharp test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	130.43	104.76	81.16	66.98	56.60	47.15	39.23	31.48	28.12	30.57
1	126.09	104.76	81.16	66.98	56.25	46.96	39.29	31.75	24.01	57.26
2	130.43	107.14	84.06	69.81	59.38	50.00	45.52	38.24	33.94	86.84
3	130.43	104.76	85.29	69.52	60.13	50.00	44.50	34.25	33.95	116.16
4	130.43	104.76	85.29	70.48	59.75	49.80	46.25	34.95	31.56	20.15
5	130.43	104.76	85.29	69.52	57.23	48.16	43.38	35.32	25.37	33.91
6	130.43	104.76	85.29	70.48	59.75	51.64	47.66	39.02	34.87	13.45

Table B.351: Average differences between CSharp production and CSharp test for the SLOC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	129.81	105.1	83.93	69.11	58.44	49.1	43.69	35.0	30.26	51.19

CC

Table B.352: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	17.0	27.0	42.0	69.0	121.0	271.0	3991.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	40.0	119.0	2354.0
Differences	100.0%	250.0%	266.67%	240.0%	285.71%	250.0%	228.57%	202.5%	127.73%	69.54%

Table B.353: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	17.0	27.0	42.0	68.0	118.0	256.0	5822.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	40.0	110.0	2774.0
Differences	100.0%	250.0%	266.67%	240.0%	285.71%	250.0%	223.81%	195.0%	132.73%	109.88%

Table B.354: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	70.0	123.0	278.0	5822.0
CSharp test CC	2.0	2.0	3.0	5.0	8.0	12.0	22.0	43.0	125.0	2992.0
Differences	100.0%	250.0%	266.67%	260.0%	237.5%	258.33%	218.18%	186.05%	122.4%	94.59%

Table B.355: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	17.0	26.0	42.0	68.0	117.0	254.0	4872.0
CSharp test CC	2.0	2.0	3.0	5.0	8.0	12.0	22.0	45.0	142.0	2972.0
Differences	100.0%	250.0%	266.67%	240.0%	225.0%	250.0%	209.09%	160.0%	78.87%	63.93%

Table B.356: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	70.0	123.0	282.0	4872.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	40.0	121.0	2354.0
Differences	100.0%	250.0%	266.67%	260.0%	285.71%	258.33%	233.33%	207.5%	133.06%	106.97%

Table B.357: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	70.0	121.0	278.0	3297.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	21.0	42.0	124.0	2972.0
Differences	100.0%	250.0%	266.67%	260.0%	285.71%	258.33%	233.33%	188.1%	124.19%	10.94%

Table B.358: Percentile values of CSharp production and CSharp test for the CC metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	4.0	7.0	11.0	18.0	27.0	43.0	70.0	123.0	279.0	5822.0
CSharp test CC	2.0	2.0	3.0	5.0	7.0	12.0	22.0	44.0	130.0	2765.0
Differences	100.0%	250.0%	266.67%	260.0%	285.71%	258.33%	218.18%	179.55%	114.62%	110.56%

Table B.359: Differences between CSharp production and CSharp test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	100.0	250.0	266.67	240.0	285.71	250.00	228.57	202.50	127.73	69.54
1	100.0	250.0	266.67	240.0	285.71	250.00	223.81	195.00	132.73	109.88
2	100.0	250.0	266.67	260.0	237.50	258.33	218.18	186.05	122.40	94.59
3	100.0	250.0	266.67	240.0	225.00	250.00	209.09	160.00	78.87	63.93
4	100.0	250.0	266.67	260.0	285.71	258.33	233.33	207.50	133.06	106.97
5	100.0	250.0	266.67	260.0	285.71	258.33	233.33	188.10	124.19	10.94
6	100.0	250.0	266.67	260.0	285.71	258.33	218.18	179.55	114.62	110.56

Table B.360: Average differences between CSharp production and CSharp test for the CC metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	100.0	250.0	266.67	251.43	270.15	254.76	223.5	188.39	119.09	80.92

Branchpoints

Table B.361: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	11.0	21.0	38.0	73.0	165.0	2408.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	18.0	85.0	2828.0
Differences	~	~	~	~	1000.0%	950.0%	533.33%	305.56%	94.12%	17.44%

Table B.362: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	22.0	39.0	75.0	167.0	2408.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	80.0	2443.0
Differences	~	~	~	~	1100.0%	1000.0%	550.0%	294.74%	108.75%	1.45%

Table B.363: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	40.0	76.0	180.0	2974.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	3.0	7.0	22.0	99.0	2828.0
Differences	~	~	~	~	1100.0%	600.0%	471.43%	245.45%	81.82%	5.16%

Table B.364: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	75.0	173.0	2974.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	19.0	91.0	2709.0
Differences	~	~	~	~	1100.0%	950.0%	550.0%	294.74%	90.11%	9.78%

Table B.365: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	11.0	21.0	39.0	74.0	165.0	1881.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	17.0	74.0	2828.0
Differences	~	~	~	~	1000.0%	950.0%	550.0%	335.29%	122.97%	50.35%

Table B.366: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	21.0	39.0	75.0	170.0	4349.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	2.0	6.0	20.0	92.0	2454.0
Differences	~	~	~	~	1100.0%	950.0%	550.0%	275.0%	84.78%	77.22%

Table B.367: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	0.0	0.0	2.0	6.0	12.0	22.0	39.0	75.0	169.0	2016.0
CSharp test Branchpoints	0.0	0.0	0.0	0.0	1.0	3.0	7.0	20.0	88.0	2709.0
Differences	~	~	~	~	1100.0%	633.33%	457.14%	275.0%	92.05%	34.38%

Table B.368: Differences between CSharp production and CSharp test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	1000.0	950.00	533.33	305.56	94.12	17.44
1	NaN	NaN	NaN	NaN	1100.0	1000.00	550.00	294.74	108.75	1.45
2	NaN	NaN	NaN	NaN	1100.0	600.00	471.43	245.45	81.82	5.16
3	NaN	NaN	NaN	NaN	1100.0	950.00	550.00	294.74	90.11	9.78
4	NaN	NaN	NaN	NaN	1000.0	950.00	550.00	335.29	122.97	50.35
5	NaN	NaN	NaN	NaN	1100.0	950.00	550.00	275.00	84.78	77.22
6	NaN	NaN	NaN	NaN	1100.0	633.33	457.14	275.00	92.05	34.38

Table B.369: Average differences between CSharp production and CSharp test for the Branchpoints metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	1071.43	861.9	523.13	289.4	96.37	27.97

Asserts

Table B.370: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	1149.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	26.0	47.0	99.0	4498.0
Differences	~	~	~	~	~	1600.0%	1200.0%	1075.0%	890.0%	291.47%

Table B.371: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	3.0	9.0	552.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	18.0	27.0	47.0	102.0	5430.0
Differences	~	~	~	~	~	1700.0%	2600.0%	1466.67%	1033.33%	883.7%

Table B.372: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	4.0	10.0	537.0
CSharp test asserts	0.0	1.0	4.0	7.0	11.0	18.0	27.0	49.0	103.0	5430.0
Differences	~	~	~	~	~	1700.0%	2600.0%	1125.0%	930.0%	911.17%

Table B.373: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	572.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	17.0	26.0	45.0	95.0	2567.0
Differences	~	~	~	~	~	1600.0%	1200.0%	1025.0%	850.0%	348.78%

Table B.374: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	4.0	10.0	1149.0
CSharp test asserts	0.0	1.0	4.0	7.0	11.0	17.0	27.0	47.0	96.0	4498.0
Differences	~	~	~	~	~	1600.0%	2600.0%	1075.0%	860.0%	291.47%

Table B.375: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	1.0	3.0	10.0	540.0
CSharp test asserts	0.0	1.0	3.0	6.0	11.0	17.0	27.0	46.0	96.0	2821.0
Differences	~	~	~	~	~	1600.0%	2600.0%	1433.33%	860.0%	422.41%

Table B.376: Percentile values of CSharp production and CSharp test for the asserts metric on a File level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	0.0	0.0	0.0	0.0	0.0	1.0	2.0	4.0	10.0	1149.0
CSharp test asserts	0.0	1.0	3.0	7.0	11.0	18.0	27.0	46.0	98.0	5430.0
Differences	~	~	~	~	~	1700.0%	1250.0%	1050.0%	880.0%	372.58%

Table B.377: Differences between CSharp production and CSharp test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	NaN	NaN	NaN	NaN	NaN	1600.0	1200.0	1075.00	890.00	291.47
1	NaN	NaN	NaN	NaN	NaN	1700.0	2600.0	1466.67	1033.33	883.70
2	NaN	NaN	NaN	NaN	NaN	1700.0	2600.0	1125.00	930.00	911.17
3	NaN	NaN	NaN	NaN	NaN	1600.0	1200.0	1025.00	850.00	348.78
4	NaN	NaN	NaN	NaN	NaN	1600.0	2600.0	1075.00	860.00	291.47
5	NaN	NaN	NaN	NaN	NaN	1600.0	2600.0	1433.33	860.00	422.41
6	NaN	NaN	NaN	NaN	NaN	1700.0	1250.0	1050.00	880.00	372.58

Table B.378: Average differences between CSharp production and CSharp test for the asserts metric on a File level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	NaN	NaN	NaN	NaN	NaN	1642.86	2007.14	1178.57	900.48	503.08

B.4.3 System level

SLOC

Table B.379: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	37910.0	79494.0	155310.0	215676.0	370024.0	587560.0	816256.0	1029293.0	1689227.0	2173399.0
CSharp test SLOC	30741.0	59540.0	100911.0	156472.0	250779.0	437622.0	487409.0	606175.0	738055.0	1319201.0
Differences	23.32%	33.51%	53.91%	37.84%	47.55%	34.26%	67.47%	69.8%	128.88%	64.75%

Table B.380: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	49418.0	139230.0	243882.0	402973.0	504964.0	639537.0	816256.0	1447269.0	1795303.0	2173399.0
CSharp test SLOC	28157.0	76196.0	141163.0	208162.0	310596.0	487409.0	571050.0	606175.0	636785.0	1594573.0
Differences	75.51%	82.73%	72.77%	93.59%	62.58%	31.21%	42.94%	138.75%	181.93%	36.3%

Table B.381: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	31388.0	62586.0	116915.0	194553.0	215676.0	370024.0	442590.0	535604.0	591154.0	1447269.0
CSharp test SLOC	32646.0	94604.0	154955.0	215156.0	333902.0	487409.0	508226.0	840121.0	1319201.0	1594573.0
Differences	4.01%	51.16%	32.54%	10.59%	54.82%	31.72%	14.83%	56.85%	123.16%	10.18%

Table B.382: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	45721.0	99098.0	195216.0	262954.0	559216.0	691704.0	859273.0	1069001.0	1689227.0	1795303.0
CSharp test SLOC	28157.0	65230.0	108472.0	250779.0	333866.0	437622.0	571050.0	626708.0	840121.0	1319201.0
Differences	62.38%	51.92%	79.97%	4.85%	67.5%	58.06%	50.47%	70.57%	101.07%	36.09%

Table B.383: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	34992.0	82256.0	155310.0	202641.0	278093.0	433541.0	559216.0	637991.0	1689227.0	2173399.0
CSharp test SLOC	29209.0	69328.0	149935.0	208162.0	333902.0	487409.0	606175.0	636785.0	1319201.0	1319201.0
Differences	19.8%	18.65%	3.58%	2.72%	20.07%	12.43%	8.4%	0.19%	28.05%	64.75%

Table B.384: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	55949.0	151184.0	235017.0	369512.0	559216.0	637991.0	816256.0	1689227.0	1795303.0	2173399.0
CSharp test SLOC	21158.0	33671.0	63029.0	121224.0	182619.0	239989.0	313855.0	383936.0	508226.0	636785.0
Differences	164.43%	349.0%	272.87%	204.82%	206.22%	165.84%	160.07%	339.98%	253.25%	241.31%

Table B.385: Percentile values of CSharp production and CSharp test for the SLOC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production SLOC	47061.0	105346.0	194945.0	268948.0	458235.0	637991.0	1029293.0	1689227.0	1795303.0	2173399.0
CSharp test SLOC	28175.0	56057.0	100625.0	150091.0	236928.0	381047.0	487409.0	606175.0	626708.0	636785.0
Differences	67.03%	87.93%	93.73%	79.19%	93.41%	67.43%	111.18%	178.67%	186.47%	241.31%

Table B.386: Differences between CSharp production and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	23.32	33.51	53.91	37.84	47.55	34.26	67.47	69.80	128.88	64.75
1	75.51	82.73	72.77	93.59	62.58	31.21	42.94	138.75	181.93	36.30
2	4.01	51.16	32.54	10.59	54.82	31.72	14.83	56.85	123.16	10.18
3	62.38	51.92	79.97	4.85	67.50	58.06	50.47	70.57	101.07	36.09
4	19.80	18.65	3.58	2.72	20.07	12.43	8.40	0.19	28.05	64.75
5	164.43	349.00	272.87	204.82	206.22	165.84	160.07	339.98	253.25	241.31
6	67.03	87.93	93.73	79.19	93.41	67.43	111.18	178.67	186.47	241.31

Table B.387: Average differences between CSharp production and CSharp test for the SLOC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	59.5	96.41	87.05	61.94	78.88	57.28	65.05	122.12	143.26	99.24

CC

Table B.388: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	10008.0	20198.0	39510.0	61448.0	83365.0	133038.0	169767.0	304054.0	307757.0	454793.0
CSharp test CC	1162.0	2592.0	5069.0	12034.0	28037.0	31757.0	32836.0	50091.0	71616.0	234164.0
Differences	761.27%	679.24%	679.44%	410.62%	197.34%	318.92%	417.01%	507.0%	329.73%	94.22%

Table B.389: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	7541.0	15870.0	30529.0	44253.0	62999.0	81631.0	114755.0	134646.0	146752.0	307757.0
CSharp test CC	1350.0	3479.0	5826.0	10999.0	22240.0	29675.0	47865.0	58190.0	80217.0	234164.0
Differences	458.59%	356.17%	424.01%	302.34%	183.27%	175.08%	139.75%	131.39%	82.94%	31.43%

Table B.390: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	10509.0	33098.0	53346.0	83365.0	114470.0	133038.0	218716.0	307757.0	346707.0	454793.0
CSharp test CC	1152.0	3050.0	5923.0	8997.0	12847.0	28037.0	31757.0	40618.0	47865.0	58190.0
Differences	812.24%	985.18%	800.66%	826.59%	791.03%	374.51%	588.72%	657.69%	624.34%	681.57%

Table B.391: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	9392.0	23500.0	45131.0	78180.0	103390.0	132217.0	218716.0	307757.0	346707.0	454793.0
CSharp test CC	1350.0	2630.0	7950.0	10999.0	28037.0	29675.0	47865.0	57269.0	71616.0	234164.0
Differences	595.7%	793.54%	467.69%	610.79%	268.76%	345.55%	356.94%	437.39%	384.12%	94.22%

Table B.392: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	8549.0	19130.0	37502.0	52958.0	76389.0	105532.0	118124.0	132217.0	218716.0	416370.0
CSharp test CC	1156.0	2592.0	6113.0	12034.0	12883.0	29675.0	47865.0	71616.0	80217.0	234164.0
Differences	639.53%	638.04%	513.48%	340.07%	492.94%	255.63%	146.79%	84.62%	172.66%	77.81%

Table B.393: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	8641.0	20198.0	32897.0	39938.0	65541.0	103390.0	133994.0	218716.0	304054.0	307757.0
CSharp test CC	1054.0	1906.0	3221.0	10374.0	12924.0	40618.0	47865.0	50091.0	57269.0	57269.0
Differences	719.83%	959.71%	921.33%	284.98%	407.13%	154.54%	179.94%	336.64%	430.92%	437.39%

Table B.394: Percentile values of CSharp production and CSharp test for the CC metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production CC	8202.0	16829.0	33098.0	59152.0	103390.0	114755.0	146752.0	218716.0	307757.0	454793.0
CSharp test CC	1540.0	3067.0	8364.0	12794.0	22240.0	29675.0	31757.0	57269.0	80217.0	234164.0
Differences	432.6%	448.71%	295.72%	362.34%	364.88%	286.71%	362.11%	281.91%	283.66%	94.22%

Table B.395: Differences between CSharp production and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	761.27	679.24	679.44	410.62	197.34	318.92	417.01	507.00	329.73	94.22
1	458.59	356.17	424.01	302.34	183.27	175.08	139.75	131.39	82.94	31.43
2	812.24	985.18	800.66	826.59	791.03	374.51	588.72	657.69	624.34	681.57
3	595.70	793.54	467.69	610.79	268.76	345.55	356.94	437.39	384.12	94.22
4	639.53	638.04	513.48	340.07	492.94	255.63	146.79	84.62	172.66	77.81
5	719.83	959.71	921.33	284.98	407.13	154.54	179.94	336.64	430.92	437.39
6	432.60	448.71	295.72	362.34	364.88	286.71	362.11	281.91	283.66	94.22

Table B.396: Average differences between CSharp production and CSharp test for the CC metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	631.39	694.37	586.05	448.25	386.48	272.99	313.04	348.09	329.77	215.84

Branchpoints

Table B.397: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	3094.0	9110.0	13918.0	21965.0	37220.0	65801.0	72772.0	88928.0	112605.0	216200.0
CSharp test Branchpoints	196.0	674.0	1404.0	2336.0	2759.0	5508.0	7598.0	20622.0	31078.0	53057.0
Differences	1478.57%	1251.63%	891.31%	840.28%	1249.04%	1094.64%	857.78%	331.23%	262.33%	307.49%

Table B.398: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	2030.0	5482.0	11089.0	18583.0	28130.0	37220.0	65801.0	72772.0	117813.0	275225.0
CSharp test Branchpoints	314.0	851.0	1505.0	2298.0	2774.0	4269.0	4963.0	9128.0	53057.0	186983.0
Differences	546.5%	544.18%	636.81%	708.66%	914.06%	771.87%	1225.83%	697.24%	122.05%	47.19%

Table B.399: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	2626.0	8053.0	11813.0	17250.0	26293.0	40252.0	66093.0	74556.0	112605.0	112605.0
CSharp test Branchpoints	243.0	721.0	1486.0	2359.0	2774.0	4377.0	7598.0	9128.0	31078.0	186983.0
Differences	980.66%	1016.92%	694.95%	631.24%	847.84%	819.63%	769.87%	716.78%	262.33%	66.05%

Table B.400: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	2360.0	6980.0	11112.0	18583.0	30834.0	39314.0	58162.0	74556.0	117813.0	243970.0
CSharp test Branchpoints	219.0	687.0	1915.0	3801.0	4494.0	7598.0	8974.0	48751.0	53057.0	186983.0
Differences	977.63%	916.01%	480.26%	388.9%	586.11%	417.43%	548.12%	52.93%	122.05%	30.48%

Table B.401: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	3112.0	10209.0	16459.0	26293.0	37430.0	69993.0	102699.0	112605.0	216200.0	275225.0
CSharp test Branchpoints	363.0	804.0	1960.0	2617.0	3286.0	4763.0	6509.0	8974.0	9188.0	53057.0
Differences	757.3%	1169.78%	739.74%	904.7%	1039.07%	1369.52%	1477.8%	1154.79%	2253.07%	418.73%

Table B.402: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	2460.0	5021.0	9110.0	11813.0	18856.0	22825.0	46383.0	75234.0	75497.0	112605.0
CSharp test Branchpoints	236.0	521.0	1410.0	2298.0	2795.0	4963.0	9128.0	31078.0	48751.0	186983.0
Differences	942.37%	863.72%	546.1%	414.06%	574.63%	359.9%	408.14%	142.08%	54.86%	66.05%

Table B.403: Percentile values of CSharp production and CSharp test for the Branchpoints metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production Branchpoints	3193.0	10782.0	18583.0	26293.0	36087.0	66093.0	75234.0	112605.0	117813.0	216200.0
CSharp test Branchpoints	289.0	1163.0	2336.0	3801.0	4963.0	9188.0	9188.0	31078.0	53057.0	186983.0
Differences	1004.84%	827.09%	695.51%	591.74%	627.12%	619.34%	718.83%	262.33%	122.05%	15.63%

Table B.404: Differences between CSharp production and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	1478.57	1251.63	891.31	840.28	1249.04	1094.64	857.78	331.23	262.33	307.49
1	546.50	544.18	636.81	708.66	914.06	771.87	1225.83	697.24	122.05	47.19
2	980.66	1016.92	694.95	631.24	847.84	819.63	769.87	716.78	262.33	66.05
3	977.63	916.01	480.26	388.90	586.11	417.43	548.12	52.93	122.05	30.48
4	757.30	1169.78	739.74	904.70	1039.07	1369.52	1477.80	1154.79	2253.07	418.73
5	942.37	863.72	546.10	414.06	574.63	359.90	408.14	142.08	54.86	66.05
6	1004.84	827.09	695.51	591.74	627.12	619.34	718.83	262.33	122.05	15.63

Table B.405: Average differences between CSharp production and CSharp test for the Branchpoints metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	955.41	941.33	669.24	639.94	833.98	778.9	858.05	479.63	456.96	135.95

Asserts

Table B.406: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (0)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	301.0	802.0	1734.0	2455.0	2899.0	4945.0	5587.0	6092.0	9748.0	14490.0
CSharp test asserts	2286.0	6191.0	10818.0	18323.0	22725.0	34168.0	41071.0	52477.0	52477.0	85742.0
Differences	659.47%	671.95%	523.88%	646.35%	683.89%	590.96%	635.12%	761.41%	438.34%	491.73%

Table B.407: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (1)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	306.0	685.0	1367.0	2598.0	3688.0	5587.0	7444.0	11033.0	14490.0	14490.0
CSharp test asserts	1593.0	4627.0	7143.0	15832.0	22725.0	41071.0	44926.0	49336.0	58844.0	58844.0
Differences	420.59%	575.47%	422.53%	509.39%	516.19%	635.12%	503.52%	347.17%	306.1%	306.1%

Table B.408: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (2)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	333.0	871.0	2071.0	2598.0	3794.0	5587.0	6907.0	10770.0	11276.0	22815.0
CSharp test asserts	1803.0	5223.0	7763.0	15832.0	20418.0	33885.0	49336.0	52477.0	58844.0	85742.0
Differences	441.44%	499.66%	274.84%	509.39%	438.17%	506.5%	614.29%	387.25%	421.85%	275.81%

Table B.409: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (3)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	306.0	685.0	1415.0	2598.0	3688.0	5479.0	7444.0	10041.0	21348.0	22815.0
CSharp test asserts	2113.0	4792.0	8011.0	12653.0	22725.0	25297.0	45135.0	50276.0	52477.0	58053.0
Differences	590.52%	599.56%	466.15%	387.03%	516.19%	361.71%	506.33%	400.71%	145.82%	154.45%

Table B.410: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (4)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	299.0	589.0	1550.0	2191.0	2640.0	3688.0	4945.0	5701.0	11033.0	11276.0
CSharp test asserts	1319.0	3820.0	6340.0	10471.0	15738.0	21865.0	29162.0	41071.0	45135.0	58844.0
Differences	341.14%	548.56%	309.03%	377.91%	496.14%	492.87%	489.73%	620.42%	309.09%	421.85%

Table B.411: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (5)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	299.0	654.0	1676.0	2372.0	2836.0	3929.0	5587.0	6907.0	8667.0	11033.0
CSharp test asserts	1622.0	4627.0	7763.0	18674.0	19682.0	41071.0	41071.0	45135.0	58844.0	58844.0
Differences	442.47%	607.49%	363.19%	687.27%	594.01%	945.33%	635.12%	553.47%	578.94%	433.35%

Table B.412: Percentile values of CSharp production and CSharp test for the asserts metric on a System level. (6)

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
CSharp production asserts	308.0	685.0	1401.0	2372.0	2655.0	3932.0	5685.0	8667.0	10041.0	10771.0
CSharp test asserts	1600.0	4626.0	8595.0	14135.0	18674.0	29171.0	41071.0	45135.0	50276.0	50276.0
Differences	419.48%	575.33%	513.49%	495.91%	603.35%	641.89%	622.45%	420.77%	400.71%	366.77%

Table B.413: Differences between CSharp production and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
0	659.47	671.95	523.88	646.35	683.89	590.96	635.12	761.41	438.34	491.73
1	420.59	575.47	422.53	509.39	516.19	635.12	503.52	347.17	306.10	306.10
2	441.44	499.66	274.84	509.39	438.17	506.50	614.29	387.25	421.85	275.81
3	590.52	599.56	466.15	387.03	516.19	361.71	506.33	400.71	145.82	154.45
4	341.14	548.56	309.03	377.91	496.14	492.87	489.73	620.42	309.09	421.85
5	442.47	607.49	363.19	687.27	594.01	945.33	635.12	553.47	578.94	433.35
6	419.48	575.33	513.49	495.91	603.35	641.89	622.45	420.77	400.71	366.77

Table B.414: Average differences between CSharp production and CSharp test for the asserts metric on a System level.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Average differences (%)	473.59	582.57	410.44	516.18	549.71	596.34	572.37	498.74	371.55	350.01

Appendix C

Validation: correlation analysis

C.1 File level

Table C.1: Pearson correlation for Java files between test code metrics and simple metrics (0)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.034366	-0.040753	-0.047207	-0.006777	0.421776
assert/CC	-0.005814	-0.012632	-0.008306	-0.003526	0.410259
Unverified Branchpoints	0.790399	0.873348	0.983178	0.066209	0.000410
Unverified CC	0.908789	0.982586	0.880500	0.073411	-0.000377

Table C.2: Coefficient of determination for Java files between test code metrics and simple metrics (0)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001181	0.001661	0.002228	0.000046	1.778948e-01
assert/CC	0.000034	0.000160	0.000069	0.000012	1.683126e-01
Unverified Branchpoints	0.624730	0.762736	0.966640	0.004384	1.684562e-07
Unverified CC	0.825898	0.965476	0.775280	0.005389	1.420383e-07

Table C.3: Pearson correlation for Java files between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.830189	-0.070112	-0.087660
assert/CC	0.830189	1.000000	-0.031201	-0.049863
Unverified Branchpoints	-0.070112	-0.031201	1.000000	0.890380
Unverified CC	-0.087660	-0.049863	0.890380	1.000000

Table C.4: Coefficient of determination for Java files between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.689214	0.004916	0.007684
assert/CC	0.689214	1.000000	0.000974	0.002486
Unverified Branchpoints	0.004916	0.000974	1.000000	0.792777
Unverified CC	0.007684	0.002486	0.792777	1.000000

Table C.5: Pearson correlation for Java files between test code metrics and simple metrics (1)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.030740	-0.039252	-0.048599	-0.008685	0.432416
assert/CC	-0.005588	-0.012563	-0.007951	-0.004383	0.394058
Unverified Branchpoints	0.777166	0.854189	0.979227	0.029480	-0.006238
Unverified CC	0.905075	0.979854	0.861148	0.041413	-0.006972

Table C.6: Coefficient of determination for Java files between test code metrics and simple metrics (1)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000945	0.001541	0.002362	0.000075	0.186983
assert/CC	0.000031	0.000158	0.000063	0.000019	0.155282
Unverified Branchpoints	0.603986	0.729638	0.958885	0.000869	0.000039
Unverified CC	0.819161	0.960113	0.741576	0.001715	0.000049

Table C.7: Pearson correlation for Java files between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.843222	-0.073701	-0.090729
assert/CC	0.843222	1.000000	-0.033122	-0.052314
Unverified Branchpoints	-0.073701	-0.033122	1.000000	0.873684
Unverified CC	-0.090729	-0.052314	0.873684	1.000000

Table C.8: Coefficient of determination for Java files between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.711023	0.005432	0.008232
assert/CC	0.711023	1.000000	0.001097	0.002737
Unverified Branchpoints	0.005432	0.001097	1.000000	0.763323
Unverified CC	0.008232	0.002737	0.763323	1.000000

Table C.9: Pearson correlation for Java files between test code metrics and simple metrics (2)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.034764	-0.042572	-0.048708	-0.006595	0.435203
assert/CC	-0.005690	-0.013679	-0.008497	-0.003465	0.399828
Unverified Branchpoints	0.812188	0.893786	0.978867	0.050224	0.000827
Unverified CC	0.895335	0.977714	0.904187	0.055745	0.000969

Table C.10: Coefficient of determination for Java files between test code metrics and simple metrics (2)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001209	0.001812	0.002373	0.000043	1.894015e-01
assert/CC	0.000032	0.000187	0.000072	0.000012	1.598625e-01
Unverified Branchpoints	0.659650	0.798853	0.958180	0.002522	6.843540e-07
Unverified CC	0.801625	0.955925	0.817555	0.003108	9.387215e-07

Table C.11: Pearson correlation for Java files between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.824981	-0.072939	-0.096182
assert/CC	0.824981	1.000000	-0.033021	-0.055274
Unverified Branchpoints	-0.072939	-0.033021	1.000000	0.916796
Unverified CC	-0.096182	-0.055274	0.916796	1.000000

Table C.12: Coefficient of determination for Java files between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.680593	0.005320	0.009251
assert/CC	0.680593	1.000000	0.001090	0.003055
Unverified Branchpoints	0.005320	0.001090	1.000000	0.840515
Unverified CC	0.009251	0.003055	0.840515	1.000000

Table C.13: Pearson correlation for Java files between test code metrics and simple metrics (3)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.028628	-0.037034	-0.045226	-0.007851	0.480206
assert/CC	-0.005053	-0.013097	-0.008359	-0.004639	0.430885
Unverified Branchpoints	0.821859	0.905387	0.982248	0.078720	-0.004786
Unverified CC	0.899737	0.980004	0.915078	0.086130	-0.005897

Table C.14: Coefficient of determination for Java files between test code metrics and simple metrics (3)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000820	0.001372	0.002045	0.000062	0.230598
assert/CC	0.000026	0.000172	0.000070	0.000022	0.185662
Unverified Branchpoints	0.675452	0.819726	0.964812	0.006197	0.000023
Unverified CC	0.809527	0.960407	0.837369	0.007418	0.000035

Table C.15: Pearson correlation for Java files between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.817542	-0.067472	-0.089037
assert/CC	0.817542	1.000000	-0.034004	-0.057026
Unverified Branchpoints	-0.067472	-0.034004	1.000000	0.926114
Unverified CC	-0.089037	-0.057026	0.926114	1.000000

Table C.16: Coefficient of determination for Java files between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.668375	0.004553	0.007927
assert/CC	0.668375	1.000000	0.001156	0.003252
Unverified Branchpoints	0.004553	0.001156	1.000000	0.857686
Unverified CC	0.007927	0.003252	0.857686	1.000000

Table C.17: Pearson correlation for Java files between test code metrics and simple metrics (4)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.033294	-0.040358	-0.048531	-0.008898	0.413793
assert/CC	-0.006198	-0.013121	-0.008386	-0.004718	0.402670
Unverified Branchpoints	0.776161	0.862889	0.980963	0.056967	0.000060
Unverified CC	0.904088	0.980718	0.870440	0.061356	-0.000910

Table C.18: Coefficient of determination for Java files between test code metrics and simple metrics (4)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001108	0.001629	0.002355	0.000079	1.712244e-01
assert/CC	0.000038	0.000172	0.000070	0.000022	1.621435e-01
Unverified Branchpoints	0.602427	0.744578	0.962288	0.003245	3.651520e-09
Unverified CC	0.817376	0.961808	0.757666	0.003765	8.274940e-07

Table C.19: Pearson correlation for Java files between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.839248	-0.073163	-0.092370
assert/CC	0.839248	1.000000	-0.033842	-0.054332
Unverified Branchpoints	-0.073163	-0.033842	1.000000	0.881479
Unverified CC	-0.092370	-0.054332	0.881479	1.000000

Table C.20: Coefficient of determination for Java files between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.704338	0.005353	0.008532
assert/CC	0.704338	1.000000	0.001145	0.002952
Unverified Branchpoints	0.005353	0.001145	1.000000	0.777006
Unverified CC	0.008532	0.002952	0.777006	1.000000

Table C.21: Pearson correlation for Java files between test code metrics and simple metrics (5)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.031908	-0.038344	-0.047574	-0.006907	0.430436
assert/CC	-0.005418	-0.012556	-0.007971	-0.003741	0.408237
Unverified Branchpoints	0.773321	0.863497	0.980410	0.049519	0.000009
Unverified CC	0.898297	0.980373	0.870150	0.057746	-0.001558

Table C.22: Coefficient of determination for Java files between test code metrics and simple metrics (5)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001018	0.001470	0.002263	0.000048	1.852752e-01
assert/CC	0.000029	0.000158	0.000064	0.000014	1.666575e-01
Unverified Branchpoints	0.598026	0.745627	0.961204	0.002452	7.862357e-11
Unverified CC	0.806938	0.961131	0.757160	0.003335	2.426879e-06

Table C.23: Pearson correlation for Java files between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.793789	-0.071078	-0.088721
assert/CC	0.793789	1.000000	-0.033332	-0.053330
Unverified Branchpoints	-0.071078	-0.033332	1.000000	0.882213
Unverified CC	-0.088721	-0.053330	0.882213	1.000000

Table C.24: Coefficient of determination for Java files between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.630101	0.005052	0.007871
assert/CC	0.630101	1.000000	0.001111	0.002844
Unverified Branchpoints	0.005052	0.001111	1.000000	0.778299
Unverified CC	0.007871	0.002844	0.778299	1.000000

Table C.25: Pearson correlation for Java files between test code metrics and simple metrics (6)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.031936	-0.040251	-0.047722	-0.006929	0.416513
assert/CC	-0.005644	-0.012469	-0.007770	-0.003505	0.387996
Unverified Branchpoints	0.769264	0.855289	0.977331	0.061315	0.002930
Unverified CC	0.900173	0.978704	0.862631	0.067192	0.003262

Table C.26: Coefficient of determination for Java files between test code metrics and simple metrics (6)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001020	0.001620	0.002277	0.000048	0.173483
assert/CC	0.000032	0.000155	0.000060	0.000012	0.150541
Unverified Branchpoints	0.591767	0.731518	0.955176	0.003760	0.000009
Unverified CC	0.810312	0.957861	0.744132	0.004515	0.000011

Table C.27: Pearson correlation for Java files between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.833203	-0.072740	-0.090404
assert/CC	0.833203	1.000000	-0.032488	-0.051466
Unverified Branchpoints	-0.072740	-0.032488	1.000000	0.875755
Unverified CC	-0.090404	-0.051466	0.875755	1.000000

Table C.28: Coefficient of determination for Java files between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.694228	0.005291	0.008173
assert/CC	0.694228	1.000000	0.001055	0.002649
Unverified Branchpoints	0.005291	0.001055	1.000000	0.766947
Unverified CC	0.008173	0.002649	0.766947	1.000000

Table C.29: Average Pearson correlations for Java files between test code metrics.

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.032234	-0.039795	-0.047652	-0.007520	0.432906
assert/CC	-0.005629	-0.012874	-0.008177	-0.003996	0.404848
Unverified Branchpoints	0.788623	0.872626	0.980318	0.056062	-0.000970
Unverified CC	0.901642	0.979993	0.880591	0.063285	-0.001640

Table C.30: Average Coefficient of determination for Java files between test code metrics.

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.001043	0.001586	0.002272	0.000057	0.187837
assert/CC	0.000032	0.000166	0.000067	0.000016	0.164066
Unverified Branchpoints	0.622291	0.761811	0.961026	0.003347	0.000010
Unverified CC	0.812977	0.960389	0.775820	0.004178	0.000014

Table C.31: Average Pearson correlations for Java files between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.826025	-0.071601	-0.090729
assert/CC	0.826025	1.000000	-0.033001	-0.053372
Unverified Branchpoints	-0.071601	-0.033001	1.000000	0.892346
Unverified CC	-0.090729	-0.053372	0.892346	1.000000

Table C.32: Average Coefficient of determination for Java files between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.682553	0.005131	0.008239
assert/CC	0.682553	1.000000	0.001090	0.002854
Unverified Branchpoints	0.005131	0.001090	1.000000	0.796650
Unverified CC	0.008239	0.002854	0.796650	1.000000

C.1.1 CSharp

Table C.33: Pearson correlation for CSharp files between test code metrics and simple metrics. (0)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.027429	-0.032701	-0.034488	-0.005547	0.553363
assert/CC	0.002312	-0.006410	-0.002187	0.003962	0.564186
Unverified Branchpoints	0.786995	0.904734	0.992554	0.192116	0.009807
Unverified CC	0.915754	0.993366	0.905289	0.258957	0.005494

Table C.34: Coefficient of determination for CSharp files between test code metrics and simple metrics. (0)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000752	0.001069	0.001189	0.000031	0.306211
assert/CC	0.000005	0.000041	0.000005	0.000016	0.318306
Unverified Branchpoints	0.619362	0.818543	0.985164	0.036909	0.000096
Unverified CC	0.838606	0.986776	0.819548	0.067059	0.000030

Table C.35: Pearson correlation for CSharp files between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.870804	-0.049418	-0.062306
assert/CC	0.870804	1.000000	-0.018674	-0.031423
Unverified Branchpoints	-0.049418	-0.018674	1.000000	0.910537
Unverified CC	-0.062306	-0.031423	0.910537	1.000000

Table C.36: Coefficient of determination for CSharp files between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.758300	0.002442	0.003882
assert/CC	0.758300	1.000000	0.000349	0.000987
Unverified Branchpoints	0.002442	0.000349	1.000000	0.829078
Unverified CC	0.003882	0.000987	0.829078	1.000000

Table C.37: Pearson correlation for CSharp files between test code metrics and simple metrics. (1)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.030322	-0.036347	-0.039015	-0.005475	0.414035
assert/CC	0.002699	-0.007781	-0.002567	0.005252	0.466891
Unverified Branchpoints	0.820761	0.919115	0.992227	0.201611	0.010197
Unverified CC	0.917335	0.993015	0.920128	0.280894	0.005613

Table C.38: Coefficient of determination for CSharp files between test code metrics and simple metrics. (1)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000919	0.001321	0.001522	0.000030	0.171425
assert/CC	0.000007	0.000061	0.000007	0.000028	0.217987
Unverified Branchpoints	0.673648	0.844773	0.984514	0.040647	0.000104
Unverified CC	0.841503	0.986079	0.846636	0.078901	0.000032

Table C.39: Pearson correlation for CSharp files between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.799964	-0.055974	-0.070202
assert/CC	0.799964	1.000000	-0.023005	-0.039001
Unverified Branchpoints	-0.055974	-0.023005	1.000000	0.925567
Unverified CC	-0.070202	-0.039001	0.925567	1.000000

Table C.40: Coefficient of determination for CSharp files between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.639943	0.003133	0.004928
assert/CC	0.639943	1.000000	0.000529	0.001521
Unverified Branchpoints	0.003133	0.000529	1.000000	0.856675
Unverified CC	0.004928	0.001521	0.856675	1.000000

Table C.41: Pearson correlation for CSharp files between test code metrics and simple metrics. (2)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.026198	-0.033029	-0.036827	-0.005292	0.406636
assert/CC	0.003909	-0.007154	-0.001930	0.005200	0.478461
Unverified Branchpoints	0.785628	0.898987	0.992579	0.199619	0.010858
Unverified CC	0.921483	0.993580	0.899366	0.309171	0.006224

Table C.42: Coefficient of determination for CSharp files between test code metrics and simple metrics. (2)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000686	0.001091	0.001356	0.000028	0.165353
assert/CC	0.000015	0.000051	0.000004	0.000027	0.228925
Unverified Branchpoints	0.617211	0.808178	0.985214	0.039848	0.000118
Unverified CC	0.849130	0.987202	0.808859	0.095587	0.000039

Table C.43: Pearson correlation for CSharp files between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.784727	-0.052995	-0.065351
assert/CC	0.784727	1.000000	-0.023040	-0.038230
Unverified Branchpoints	-0.052995	-0.023040	1.000000	0.904521
Unverified CC	-0.065351	-0.038230	0.904521	1.000000

Table C.44: Coefficient of determination for CSharp files between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.615797	0.002808	0.004271
assert/CC	0.615797	1.000000	0.000531	0.001462
Unverified Branchpoints	0.002808	0.000531	1.000000	0.818158
Unverified CC	0.004271	0.001462	0.818158	1.000000

Table C.45: Pearson correlation for CSharp files between test code metrics and simple metrics. (3)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.021756	-0.031350	-0.035006	-0.005251	0.558953
assert/CC	0.003714	-0.006282	-0.002251	0.003133	0.572144
Unverified Branchpoints	0.772680	0.894778	0.992615	0.227070	0.009926
Unverified CC	0.921655	0.994015	0.893933	0.348290	0.004549

Table C.46: Coefficient of determination for CSharp files between test code metrics and simple metrics. (3)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000473	0.000983	0.001225	0.000028	0.312429
assert/CC	0.000014	0.000039	0.000005	0.000010	0.327348
Unverified Branchpoints	0.597034	0.800627	0.985285	0.051561	0.000099
Unverified CC	0.849448	0.988067	0.799117	0.121306	0.000021

Table C.47: Pearson correlation for CSharp files between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.868863	-0.050211	-0.059490
assert/CC	0.868863	1.000000	-0.018910	-0.030341
Unverified Branchpoints	-0.050211	-0.018910	1.000000	0.899542
Unverified CC	-0.059490	-0.030341	0.899542	1.000000

Table C.48: Coefficient of determination for CSharp files between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.754922	0.002521	0.003539
assert/CC	0.754922	1.000000	0.000358	0.000921
Unverified Branchpoints	0.002521	0.000358	1.000000	0.809176
Unverified CC	0.003539	0.000921	0.809176	1.000000

Table C.49: Pearson correlation for CSharp files between test code metrics and simple metrics. (4)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.030922	-0.039165	-0.040795	-0.006296	0.443030
assert/CC	0.003061	-0.008009	-0.002841	0.004860	0.493449
Unverified Branchpoints	0.810335	0.914959	0.992110	0.210186	0.011288
Unverified CC	0.909481	0.993030	0.916000	0.296815	0.006830

Table C.50: Coefficient of determination for CSharp files between test code metrics and simple metrics. (4)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000956	0.001534	0.001664	0.000040	0.196275
assert/CC	0.000009	0.000064	0.000008	0.000024	0.243492
Unverified Branchpoints	0.656642	0.837151	0.984282	0.044178	0.000127
Unverified CC	0.827156	0.986108	0.839056	0.088099	0.000047

Table C.51: Pearson correlation for CSharp files between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.840258	-0.058009	-0.072009
assert/CC	0.840258	1.000000	-0.022449	-0.037794
Unverified Branchpoints	-0.058009	-0.022449	1.000000	0.921406
Unverified CC	-0.072009	-0.037794	0.921406	1.000000

Table C.52: Coefficient of determination for CSharp files between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.706033	0.003365	0.005185
assert/CC	0.706033	1.000000	0.000504	0.001428
Unverified Branchpoints	0.003365	0.000504	1.000000	0.848990
Unverified CC	0.005185	0.001428	0.848990	1.000000

Table C.53: Pearson correlation for CSharp files between test code metrics and simple metrics. (5)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.023527	-0.033656	-0.035652	-0.005354	0.548935
assert/CC	0.003896	-0.006424	-0.002056	0.003562	0.563718
Unverified Branchpoints	0.795254	0.897713	0.991541	0.201198	0.011137
Unverified CC	0.927887	0.993080	0.897013	0.307160	0.005828

Table C.54: Coefficient of determination for CSharp files between test code metrics and simple metrics. (5)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000554	0.001133	0.001271	0.000029	0.301329
assert/CC	0.000015	0.000041	0.000004	0.000013	0.317779
Unverified Branchpoints	0.632429	0.805888	0.983153	0.040481	0.000124
Unverified CC	0.860974	0.986208	0.804632	0.094348	0.000034

Table C.55: Pearson correlation for CSharp files between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.880118	-0.051279	-0.061934
assert/CC	0.880118	1.000000	-0.019756	-0.032073
Unverified Branchpoints	-0.051279	-0.019756	1.000000	0.903391
Unverified CC	-0.061934	-0.032073	0.903391	1.000000

Table C.56: Coefficient of determination for CSharp files between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.774607	0.002630	0.003836
assert/CC	0.774607	1.000000	0.000390	0.001029
Unverified Branchpoints	0.002630	0.000390	1.000000	0.816115
Unverified CC	0.003836	0.001029	0.816115	1.000000

Table C.57: Pearson correlation for CSharp files between test code metrics and simple metrics. (6)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.027084	-0.034926	-0.038554	-0.006465	0.434301
assert/CC	0.003361	-0.006856	-0.002252	0.003367	0.484514
Unverified Branchpoints	0.776303	0.896872	0.992369	0.242699	0.016936
Unverified CC	0.924859	0.993806	0.896145	0.369794	0.010805

Table C.58: Coefficient of determination for CSharp files between test code metrics and simple metrics. (6)

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000734	0.001220	0.001486	0.000042	0.188617
assert/CC	0.000011	0.000047	0.000005	0.000011	0.234754
Unverified Branchpoints	0.602647	0.804379	0.984797	0.058903	0.000287
Unverified CC	0.855363	0.987650	0.803076	0.136748	0.000117

Table C.59: Pearson correlation for CSharp files between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.827612	-0.055506	-0.066374
assert/CC	0.827612	1.000000	-0.021266	-0.034358
Unverified Branchpoints	-0.055506	-0.021266	1.000000	0.901790
Unverified CC	-0.066374	-0.034358	0.901790	1.000000

Table C.60: Coefficient of determination for CSharp files between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.684942	0.003081	0.004406
assert/CC	0.684942	1.000000	0.000452	0.001180
Unverified Branchpoints	0.003081	0.000452	1.000000	0.813226
Unverified CC	0.004406	0.001180	0.813226	1.000000

Table C.61: Average Pearson correlations for CSharp files between test code metrics.

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	-0.026748	-0.034453	-0.037191	-0.005668	0.479893
assert/CC	0.003279	-0.006988	-0.002298	0.004191	0.517623
Unverified Branchpoints	0.792565	0.903880	0.992285	0.210643	0.011450
Unverified CC	0.919779	0.993413	0.903982	0.310155	0.006478

Table C.62: Average Coefficient of determination for CSharp files between test code metrics.

	SLOC	CC	Branchpoints	asserts	Direct asserts
assert/branch	0.000725	0.001193	0.001388	0.000032	0.234520
assert/CC	0.000011	0.000049	0.000005	0.000018	0.269799
Unverified Branchpoints	0.628425	0.817077	0.984630	0.044647	0.000136
Unverified CC	0.846026	0.986870	0.817275	0.097435	0.000045

Table C.63: Average Pearson correlations for CSharp files between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.838907	-0.053342	-0.065381
assert/CC	0.838907	1.000000	-0.021014	-0.034746
Unverified Branchpoints	-0.053342	-0.021014	1.000000	0.909536
Unverified CC	-0.065381	-0.034746	0.909536	1.000000

Table C.64: Average Coefficient of determination for CSharp files between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC
assert/branch	1.000000	0.704935	0.002854	0.004292
assert/CC	0.704935	1.000000	0.000445	0.001218
Unverified Branchpoints	0.002854	0.000445	1.000000	0.827345
Unverified CC	0.004292	0.001218	0.827345	1.000000

C.2 System level

C.2.1 Java

Table C.65: Pearson correlation for Java systems between test code metrics and simple metrics (0)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.123278	0.025515	-0.125482	-0.138161	-0.012191	0.051350	0.122692
assert/CC	-0.085243	0.078954	-0.087352	-0.096904	0.029357	0.118049	0.199347
Unverified Branchpoints	0.803334	0.339058	0.849539	0.917456	0.159497	0.308106	0.156246
Unverified CC	0.963943	0.619041	0.981341	0.975442	0.327045	0.591610	0.458105
test/prod	-0.026704	0.002286	-0.025543	-0.022721	-0.006611	-0.017951	-0.015071

Table C.66: Coefficient of determination for Java systems between test code metrics and simple metrics (0)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.015198	0.000651	0.015746	0.019089	0.000149	0.002637	0.015053
assert/CC	0.007266	0.006234	0.007630	0.009390	0.000862	0.013936	0.039739
Unverified Branchpoints	0.645345	0.114960	0.721717	0.841725	0.025439	0.094929	0.024413
Unverified CC	0.929186	0.383212	0.963030	0.951486	0.106958	0.350002	0.209860
test/prod	0.000713	0.000005	0.000652	0.000516	0.000044	0.000322	0.000227

Table C.67: Pearson correlation for Java systems between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.870042	-0.194022	-0.171895	-0.043408
assert/CC	0.870042	1.000000	-0.184265	-0.147366	-0.047763
Unverified Branchpoints	-0.194022	-0.184265	1.000000	0.924671	-0.019938
Unverified CC	-0.171895	-0.147366	0.924671	1.000000	-0.025262
test/prod	-0.043408	-0.047763	-0.019938	-0.025262	1.000000

Table C.68: Coefficient of determination for Java systems between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.756973	0.037645	0.029548	0.001884
assert/CC	0.756973	1.000000	0.033954	0.021717	0.002281
Unverified Branchpoints	0.037645	0.033954	1.000000	0.855017	0.000398
Unverified CC	0.029548	0.021717	0.855017	1.000000	0.000638
test/prod	0.001884	0.002281	0.000398	0.000638	1.000000

Table C.69: Pearson correlation for Java systems between test code metrics and simple metrics (1)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.157780	-0.009145	-0.161131	-0.169988	-0.054703	0.010983	0.087815
assert/CC	-0.129695	0.025202	-0.132517	-0.142994	-0.052677	0.055197	0.139505
Unverified Branchpoints	0.822897	0.337187	0.861424	0.941891	0.179956	0.326825	0.158060
Unverified CC	0.971970	0.609437	0.985749	0.984503	0.280521	0.609156	0.471711
test/prod	-0.036356	0.004712	-0.034568	-0.030017	-0.012603	-0.024851	-0.020896

Table C.70: Coefficient of determination for Java systems between test code metrics and simple metrics (1)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.024895	0.000084	0.025963	0.028896	0.002992	0.000121	0.007711
assert/CC	0.016821	0.000635	0.017561	0.020447	0.002775	0.003047	0.019462
Unverified Branchpoints	0.677159	0.113695	0.742051	0.887158	0.032384	0.106815	0.024983
Unverified CC	0.944725	0.371413	0.971702	0.969246	0.078692	0.371071	0.222511
test/prod	0.001322	0.000022	0.001195	0.000901	0.000159	0.000618	0.000437

Table C.71: Pearson correlation for Java systems between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.899021	-0.208259	-0.198532	-0.047165
assert/CC	0.899021	1.000000	-0.198493	-0.177559	-0.065002
Unverified Branchpoints	-0.208259	-0.198493	1.000000	0.928014	-0.026310
Unverified CC	-0.198532	-0.177559	0.928014	1.000000	-0.034142
test/prod	-0.047165	-0.065002	-0.026310	-0.034142	1.000000

Table C.72: Coefficient of determination for Java systems between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.808239	0.043372	0.039415	0.002225
assert/CC	0.808239	1.000000	0.039399	0.031527	0.004225
Unverified Branchpoints	0.043372	0.039399	1.000000	0.861209	0.000692
Unverified CC	0.039415	0.031527	0.861209	1.000000	0.001166
test/prod	0.002225	0.004225	0.000692	0.001166	1.000000

Table C.73: Pearson correlation for Java systems between test code metrics and simple metrics (2)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.176828	-0.001195	-0.180316	-0.181782	-0.024066	0.023303	0.104639
assert/CC	-0.147529	0.062600	-0.153067	-0.158803	0.003909	0.097135	0.192757
Unverified Branchpoints	0.867626	0.233797	0.907612	0.957597	0.113551	0.204600	0.073063
Unverified CC	0.971115	0.449622	0.987134	0.980162	0.227532	0.422451	0.292229
test/prod	-0.091090	0.144236	-0.095930	-0.104378	0.010955	0.107875	0.090194

Table C.74: Coefficient of determination for Java systems between test code metrics and simple metrics (2)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.031268	0.000001	0.032514	0.033045	0.000579	0.000543	0.010949
assert/CC	0.021765	0.003919	0.023430	0.025219	0.000015	0.009435	0.037155
Unverified Branchpoints	0.752774	0.054661	0.823759	0.916991	0.012894	0.041861	0.005338
Unverified CC	0.943065	0.202160	0.974433	0.960717	0.051771	0.178465	0.085398
test/prod	0.008297	0.020804	0.009203	0.010895	0.000120	0.011637	0.008135

Table C.75: Pearson correlation for Java systems between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.880600	-0.212403	-0.210781	0.375059
assert/CC	0.880600	1.000000	-0.214088	-0.197414	0.356762
Unverified Branchpoints	-0.212403	-0.214088	1.000000	0.954303	-0.133055
Unverified CC	-0.210781	-0.197414	0.954303	1.000000	-0.118292
test/prod	0.375059	0.356762	-0.133055	-0.118292	1.000000

Table C.76: Coefficient of determination for Java systems between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.775457	0.045115	0.044429	0.140669
assert/CC	0.775457	1.000000	0.045834	0.038972	0.127279
Unverified Branchpoints	0.045115	0.045834	1.000000	0.910693	0.017704
Unverified CC	0.044429	0.038972	0.910693	1.000000	0.013993
test/prod	0.140669	0.127279	0.017704	0.013993	1.000000

Table C.77: Pearson correlation for Java systems between test code metrics and simple metrics (3)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.143860	0.019520	-0.147030	-0.153751	-0.027437	0.024166	0.067170
assert/CC	-0.103875	0.083890	-0.108451	-0.118645	0.002617	0.099705	0.149948
Unverified Branchpoints	0.827791	0.165306	0.866884	0.930940	0.154735	0.157617	0.092776
Unverified CC	0.965847	0.458933	0.981963	0.990294	0.298694	0.453915	0.391495
test/prod	-0.038271	0.001960	-0.036488	-0.031608	-0.009398	-0.025746	-0.022039

Table C.78: Coefficient of determination for Java systems between test code metrics and simple metrics (3)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.020696	0.000381	0.021618	0.023639	0.000753	0.000584	0.004512
assert/CC	0.010790	0.007038	0.011762	0.014077	0.000007	0.009941	0.022484
Unverified Branchpoints	0.685238	0.027326	0.751488	0.866650	0.023943	0.024843	0.008607
Unverified CC	0.932861	0.210619	0.964251	0.980682	0.089218	0.206039	0.153268
test/prod	0.001465	0.000004	0.001331	0.000999	0.000088	0.000663	0.000486

Table C.79: Pearson correlation for Java systems between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.843818	-0.185168	-0.178343	-0.045749
assert/CC	0.843818	1.000000	-0.179395	-0.154386	-0.076097
Unverified Branchpoints	-0.185168	-0.179395	1.000000	0.940439	-0.026675
Unverified CC	-0.178343	-0.154386	0.940439	1.000000	-0.035453
test/prod	-0.045749	-0.076097	-0.026675	-0.035453	1.000000

Table C.80: Coefficient of determination for Java systems between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.712028	0.034287	0.031806	0.002093
assert/CC	0.712028	1.000000	0.032183	0.023835	0.005791
Unverified Branchpoints	0.034287	0.032183	1.000000	0.884425	0.000712
Unverified CC	0.031806	0.023835	0.884425	1.000000	0.001257
test/prod	0.002093	0.005791	0.000712	0.001257	1.000000

Table C.81: Pearson correlation for Java systems between test code metrics and simple metrics (4)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.195239	-0.012602	-0.192240	-0.179996	-0.088068	0.034648	0.183385
assert/CC	-0.199286	0.007262	-0.195424	-0.179464	-0.094862	0.076700	0.261738
Unverified Branchpoints	0.925625	0.301090	0.961465	0.995034	0.126844	0.281714	0.077050
Unverified CC	0.983780	0.413947	0.996576	0.972414	0.171486	0.372102	0.135606
test/prod	-0.069406	0.127509	-0.068286	-0.064611	-0.009551	0.033479	0.043420

Table C.82: Coefficient of determination for Java systems between test code metrics and simple metrics (4)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.038118	0.000159	0.036956	0.032398	0.007756	0.001201	0.033630
assert/CC	0.039715	0.000053	0.038191	0.032207	0.008999	0.005883	0.068507
Unverified Branchpoints	0.856781	0.090655	0.924414	0.990093	0.016090	0.079363	0.005937
Unverified CC	0.967823	0.171352	0.993163	0.945590	0.029407	0.138460	0.018389
test/prod	0.004817	0.016259	0.004663	0.004175	0.000091	0.001121	0.001885

Table C.83: Pearson correlation for Java systems between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.879420	-0.191880	-0.210594	0.144835
assert/CC	0.879420	1.000000	-0.199897	-0.220449	0.136679
Unverified Branchpoints	-0.191880	-0.199897	1.000000	0.969326	-0.068226
Unverified CC	-0.210594	-0.220449	0.969326	1.000000	-0.072977
test/prod	0.144835	0.136679	-0.068226	-0.072977	1.000000

Table C.84: Coefficient of determination for Java systems between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.773380	0.036818	0.044350	0.020977
assert/CC	0.773380	1.000000	0.039959	0.048598	0.018681
Unverified Branchpoints	0.036818	0.039959	1.000000	0.939593	0.004655
Unverified CC	0.044350	0.048598	0.939593	1.000000	0.005326
test/prod	0.020977	0.018681	0.004655	0.005326	1.000000

Table C.85: Pearson correlation for Java systems between test code metrics and simple metrics (5)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.128973	0.017608	-0.132995	-0.144730	-0.004043	0.051534	0.130731
assert/CC	-0.117516	0.035874	-0.121733	-0.130958	0.021785	0.083768	0.172885
Unverified Branchpoints	0.838713	0.257423	0.890296	0.963434	0.105319	0.246766	0.114297
Unverified CC	0.970095	0.518017	0.988248	0.982637	0.261327	0.515327	0.383361
test/prod	-0.031843	0.009839	-0.030602	-0.026423	-0.005558	-0.021542	-0.019057

Table C.86: Coefficient of determination for Java systems between test code metrics and simple metrics (5)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.016634	0.000310	0.017688	0.020947	0.000016	0.002656	0.017091
assert/CC	0.013810	0.001287	0.014819	0.017150	0.000475	0.007017	0.029889
Unverified Branchpoints	0.703439	0.066267	0.792627	0.928205	0.011092	0.060893	0.013064
Unverified CC	0.941085	0.268341	0.976634	0.965576	0.068292	0.265562	0.146966
test/prod	0.001014	0.000097	0.000937	0.000698	0.000031	0.000464	0.000363

Table C.87: Pearson correlation for Java systems between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.892861	-0.173826	-0.167152	-0.041351
assert/CC	0.892861	1.000000	-0.173089	-0.162485	-0.058167
Unverified Branchpoints	-0.173826	-0.173089	1.000000	0.942240	-0.023164
Unverified CC	-0.167152	-0.162485	0.942240	1.000000	-0.029682
test/prod	-0.041351	-0.058167	-0.023164	-0.029682	1.000000

Table C.88: Coefficient of determination for Java systems between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.797201	0.030215	0.027940	0.001710
assert/CC	0.797201	1.000000	0.029960	0.026401	0.003383
Unverified Branchpoints	0.030215	0.029960	1.000000	0.887816	0.000537
Unverified CC	0.027940	0.026401	0.887816	1.000000	0.000881
test/prod	0.001710	0.003383	0.000537	0.000881	1.000000

Table C.89: Pearson correlation for Java systems between test code metrics and simple metrics (6)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.149475	0.072069	-0.151430	-0.157073	-0.024492	0.094615	0.155283
assert/CC	-0.130445	0.128051	-0.133340	-0.138860	0.004321	0.159250	0.226744
Unverified Branchpoints	0.840340	0.214325	0.891428	0.959512	0.163951	0.204789	0.108870
Unverified CC	0.968282	0.471865	0.987245	0.971608	0.305940	0.439707	0.341212
test/prod	-0.028431	0.013660	-0.026980	-0.023344	-0.006292	-0.017297	-0.014970

Table C.90: Coefficient of determination for Java systems between test code metrics and simple metrics (6)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.022343	0.005194	0.022931	0.024672	0.000600	0.008952	0.024113
assert/CC	0.017016	0.016397	0.017780	0.019282	0.000019	0.025361	0.051413
Unverified Branchpoints	0.706171	0.045935	0.794644	0.920664	0.026880	0.041939	0.011853
Unverified CC	0.937570	0.222656	0.974653	0.944022	0.093600	0.193342	0.116426
test/prod	0.000808	0.000187	0.000728	0.000545	0.000040	0.000299	0.000224

Table C.91: Pearson correlation for Java systems between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.865744	-0.193429	-0.191171	-0.040961
assert/CC	0.865744	1.000000	-0.195745	-0.184688	-0.046130
Unverified Branchpoints	-0.193429	-0.195745	1.000000	0.939253	-0.020757
Unverified CC	-0.191171	-0.184688	0.939253	1.000000	-0.026301
test/prod	-0.040961	-0.046130	-0.020757	-0.026301	1.000000

Table C.92: Coefficient of determination for Java systems between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.749513	0.037415	0.036546	0.001678
assert/CC	0.749513	1.000000	0.038316	0.034110	0.002128
Unverified Branchpoints	0.037415	0.038316	1.000000	0.882197	0.000431
Unverified CC	0.036546	0.034110	0.882197	1.000000	0.000692
test/prod	0.001678	0.002128	0.000431	0.000692	1.000000

Table C.93: Average Pearson correlations for Java systems between test code metrics.

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.153634	0.015967	-0.155803	-0.160783	-0.033571	0.041514	0.121674
assert/CC	-0.130513	0.060262	-0.133126	-0.138090	-0.012221	0.098543	0.191846
test/prod	-0.046014	0.043457	-0.045485	-0.043300	-0.005580	0.004853	0.005940
Unverified Branchpoints	0.846618	0.264026	0.889807	0.952266	0.143408	0.247202	0.111480
Unverified CC	0.970719	0.505837	0.986894	0.979580	0.267506	0.486324	0.353388

Table C.94: Average Coefficient of determination for Java systems between test code metrics.

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.024164	0.000969	0.024774	0.026098	0.001835	0.002385	0.016151
assert/CC	0.018169	0.005080	0.018739	0.019682	0.001879	0.010660	0.038378
test/prod	0.002634	0.005340	0.002673	0.002676	0.000082	0.002161	0.001680
Unverified Branchpoints	0.718130	0.073357	0.792957	0.907355	0.021246	0.064378	0.013456
Unverified CC	0.942331	0.261393	0.973981	0.959617	0.073991	0.243277	0.136117

Table C.95: Average Pearson correlations for Java systems between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.875929	-0.194141	-0.189781	0.043037
assert/CC	0.875929	1.000000	-0.192139	-0.177764	0.028612
test/prod	0.043037	0.028612	-0.045446	-0.048873	1.000000
Unverified Branchpoints	-0.194141	-0.192139	1.000000	0.942607	-0.045446
Unverified CC	-0.189781	-0.177764	0.942607	1.000000	-0.048873

Table C.96: Average Coefficient of determination for Java systems between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.767542	0.037838	0.036291	0.024462
assert/CC	0.767542	1.000000	0.037086	0.032166	0.023396
test/prod	0.024462	0.023396	0.003590	0.003422	1.000000
Unverified Branchpoints	0.037838	0.037086	1.000000	0.888707	0.003590
Unverified CC	0.036291	0.032166	0.888707	1.000000	0.003422

C.2.2 CSharp

Table C.97: Pearson correlation for CSharp systems between test code metrics and simple metrics (0)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.064379	0.079515	-0.065895	-0.073712	-0.044274	0.146857	0.252945
assert/CC	-0.037973	0.053591	-0.039797	-0.042581	-0.021671	0.114070	0.195761
Unverified Branchpoints	0.928394	0.498630	0.937649	0.989378	0.811422	0.318258	0.178447
Unverified CC	0.989840	0.640416	0.994785	0.973724	0.877642	0.493722	0.310425
test/prod	-0.063111	0.097345	-0.062121	-0.068022	-0.053718	0.096887	0.075662

Table C.98: Coefficient of determination for CSharp systems between test code metrics and simple metrics (0)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.004145	0.006323	0.004342	0.005434	0.001960	0.021567	0.063981
assert/CC	0.001442	0.002872	0.001584	0.001813	0.000470	0.013012	0.038323
Unverified Branchpoints	0.861916	0.248632	0.879185	0.978869	0.658406	0.101288	0.031843
Unverified CC	0.979783	0.410132	0.989597	0.948138	0.770256	0.243762	0.096364
test/prod	0.003983	0.009476	0.003859	0.004627	0.002886	0.009387	0.005725

Table C.99: Pearson correlation for CSharp systems between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.907802	-0.096235	-0.093966	0.152286
assert/CC	0.907802	1.000000	-0.060960	-0.059539	0.116628
Unverified Branchpoints	-0.096235	-0.060960	1.000000	0.955333	-0.077235
Unverified CC	-0.093966	-0.059539	0.955333	1.000000	-0.072859
test/prod	0.152286	0.116628	-0.077235	-0.072859	1.000000

Table C.100: Coefficient of determination for CSharp systems between test code metrics (0)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.824105	0.009261	0.008830	0.023191
assert/CC	0.824105	1.000000	0.003716	0.003545	0.013602
Unverified Branchpoints	0.009261	0.003716	1.000000	0.912661	0.005965
Unverified CC	0.008830	0.003545	0.912661	1.000000	0.005308
test/prod	0.023191	0.013602	0.005965	0.005308	1.000000

Table C.101: Pearson correlation for CSharp systems between test code metrics and simple metrics (1)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.060646	0.041295	-0.064690	-0.065292	-0.032952	0.133478	0.236736
assert/CC	-0.034897	0.040419	-0.038581	-0.036359	-0.011062	0.129785	0.227050
Unverified Branchpoints	0.932090	0.731818	0.937852	0.985634	0.896262	0.480721	0.159275
Unverified CC	0.992583	0.813425	0.995697	0.975704	0.942679	0.635152	0.306114
test/prod	0.005304	0.261186	0.005600	-0.022824	0.038489	0.319548	0.262201

Table C.102: Coefficient of determination for CSharp systems between test code metrics and simple metrics (1)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.003678	0.001705	0.004185	0.004263	0.001086	0.017816	0.056044
assert/CC	0.001218	0.001634	0.001489	0.001322	0.000122	0.016844	0.051552
Unverified Branchpoints	0.868792	0.535558	0.879565	0.971475	0.803286	0.231093	0.025369
Unverified CC	0.985221	0.661661	0.991413	0.951999	0.888644	0.403418	0.093706
test/prod	0.000028	0.068218	0.000031	0.000521	0.001481	0.102111	0.068749

Table C.103: Pearson correlation for CSharp systems between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.922406	-0.087910	-0.086167	0.486754
assert/CC	0.922406	1.000000	-0.056942	-0.057556	0.378490
Unverified Branchpoints	-0.087910	-0.056942	1.000000	0.956637	-0.052810
Unverified CC	-0.086167	-0.057556	0.956637	1.000000	-0.017883
test/prod	0.486754	0.378490	-0.052810	-0.017883	1.000000

Table C.104: Coefficient of determination for CSharp systems between test code metrics (1)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.850834	0.007728	0.007425	0.236929
assert/CC	0.850834	1.000000	0.003242	0.003313	0.143255
Unverified Branchpoints	0.007728	0.003242	1.000000	0.915155	0.002789
Unverified CC	0.007425	0.003313	0.915155	1.000000	0.000320
test/prod	0.236929	0.143255	0.002789	0.000320	1.000000

Table C.105: Pearson correlation for CSharp systems between test code metrics and simple metrics (2)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.065471	0.152310	-0.068592	-0.077281	-0.010512	0.219211	0.267947
assert/CC	-0.032485	0.206402	-0.037037	-0.043963	0.037848	0.310651	0.379370
Unverified Branchpoints	0.881984	0.442846	0.892302	0.973005	0.760740	0.308122	0.239092
Unverified CC	0.990197	0.683607	0.994099	0.968609	0.891107	0.568596	0.475774
test/prod	-0.061725	0.091221	-0.060993	-0.066906	-0.037405	0.098759	0.091240

Table C.106: Coefficient of determination for CSharp systems between test code metrics and simple metrics (2)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.004286	0.023198	0.004705	0.005972	0.000111	0.048053	0.071796
assert/CC	0.001055	0.042602	0.001372	0.001933	0.001432	0.096504	0.143921
Unverified Branchpoints	0.777896	0.196113	0.796202	0.946739	0.578726	0.094939	0.057165
Unverified CC	0.980491	0.467318	0.988233	0.938203	0.794072	0.323301	0.226360
test/prod	0.003810	0.008321	0.003720	0.004476	0.001399	0.009753	0.008325

Table C.107: Pearson correlation for CSharp systems between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.913526	-0.121190	-0.105012	0.258097
assert/CC	0.913526	1.000000	-0.104292	-0.084404	0.260472
Unverified Branchpoints	-0.121190	-0.104292	1.000000	0.924565	-0.086188
Unverified CC	-0.105012	-0.084404	0.924565	1.000000	-0.076402
test/prod	0.258097	0.260472	-0.086188	-0.076402	1.000000

Table C.108: Coefficient of determination for CSharp systems between test code metrics (2)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.834530	0.014687	0.011028	0.066614
assert/CC	0.834530	1.000000	0.010877	0.007124	0.067846
Unverified Branchpoints	0.014687	0.010877	1.000000	0.854820	0.007428
Unverified CC	0.011028	0.007124	0.854820	1.000000	0.005837
test/prod	0.066614	0.067846	0.007428	0.005837	1.000000

Table C.109: Pearson correlation for CSharp systems between test code metrics and simple metrics (3)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.069686	0.062023	-0.070985	-0.071195	-0.044781	0.132298	0.218468
assert/CC	-0.040702	0.063719	-0.041953	-0.040008	-0.018668	0.135797	0.215581
Unverified Branchpoints	0.919977	0.535214	0.937644	0.988228	0.794037	0.360582	0.222191
Unverified CC	0.992245	0.661199	0.996743	0.968125	0.913550	0.509740	0.353322
test/prod	-0.055130	0.110226	-0.053827	-0.057327	-0.037592	0.128100	0.099130

Table C.110: Coefficient of determination for CSharp systems between test code metrics and simple metrics (3)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.004856	0.003847	0.005039	0.005069	0.002005	0.017503	0.047728
assert/CC	0.001657	0.004060	0.001760	0.001601	0.000349	0.018441	0.046475
Unverified Branchpoints	0.846357	0.286454	0.879176	0.976595	0.630494	0.130019	0.049369
Unverified CC	0.984551	0.437185	0.993497	0.937266	0.834574	0.259835	0.124837
test/prod	0.003039	0.012150	0.002897	0.003286	0.001413	0.016410	0.009827

Table C.111: Pearson correlation for CSharp systems between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.911547	-0.088406	-0.089375	0.237190
assert/CC	0.911547	1.000000	-0.056966	-0.058284	0.179744
Unverified Branchpoints	-0.088406	-0.056966	1.000000	0.951769	-0.067484
Unverified CC	-0.089375	-0.058284	0.951769	1.000000	-0.063708
test/prod	0.237190	0.179744	-0.067484	-0.063708	1.000000

Table C.112: Coefficient of determination for CSharp systems between test code metrics (3)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.830917	0.007816	0.007988	0.056259
assert/CC	0.830917	1.000000	0.003245	0.003397	0.032308
Unverified Branchpoints	0.007816	0.003245	1.000000	0.905865	0.004554
Unverified CC	0.007988	0.003397	0.905865	1.000000	0.004059
test/prod	0.056259	0.032308	0.004554	0.004059	1.000000

Table C.113: Pearson correlation for CSharp systems between test code metrics and simple metrics (4)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.071533	0.017202	-0.072667	-0.071970	-0.060783	0.088003	0.203443
assert/CC	-0.051575	0.046872	-0.052805	-0.055901	-0.033584	0.156533	0.305426
Unverified Branchpoints	0.958686	0.779633	0.965867	0.994964	0.905779	0.587618	0.244928
Unverified CC	0.996033	0.832409	0.998517	0.986388	0.936876	0.694641	0.388798
test/prod	-0.029252	0.080972	-0.030571	-0.028661	-0.016299	0.099580	0.076769

Table C.114: Coefficient of determination for CSharp systems between test code metrics and simple metrics (4)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.005117	0.000296	0.005280	0.005180	0.003695	0.007745	0.041389
assert/CC	0.002660	0.002197	0.002788	0.003125	0.001128	0.024503	0.093285
Unverified Branchpoints	0.919079	0.607828	0.932899	0.989953	0.820435	0.345294	0.059990
Unverified CC	0.992081	0.692906	0.997036	0.972962	0.877737	0.482526	0.151164
test/prod	0.000856	0.006557	0.000935	0.000821	0.000266	0.009916	0.005893

Table C.115: Pearson correlation for CSharp systems between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.949012	-0.086536	-0.086812	0.250550
assert/CC	0.949012	1.000000	-0.080569	-0.072630	0.263138
Unverified Branchpoints	-0.086536	-0.080569	1.000000	0.975246	-0.034507
Unverified CC	-0.086812	-0.072630	0.975246	1.000000	-0.035988
test/prod	0.250550	0.263138	-0.034507	-0.035988	1.000000

Table C.116: Coefficient of determination for CSharp systems between test code metrics (4)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.900623	0.007489	0.007536	0.062775
assert/CC	0.900623	1.000000	0.006491	0.005275	0.069242
Unverified Branchpoints	0.007489	0.006491	1.000000	0.951105	0.001191
Unverified CC	0.007536	0.005275	0.951105	1.000000	0.001295
test/prod	0.062775	0.069242	0.001191	0.001295	1.000000

Table C.117: Pearson correlation for CSharp systems between test code metrics and simple metrics (5)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.037600	0.146023	-0.040310	-0.052257	0.008530	0.216902	0.265160
assert/CC	-0.015096	0.118229	-0.018063	-0.024674	0.023848	0.189049	0.226531
Unverified Branchpoints	0.891324	0.506617	0.896220	0.972992	0.786465	0.331434	0.231582
Unverified CC	0.987374	0.697261	0.990905	0.975740	0.878795	0.548557	0.435054
test/prod	-0.040165	0.130769	-0.038086	-0.044867	-0.018821	0.131318	0.111481

Table C.118: Coefficient of determination for CSharp systems between test code metrics and simple metrics (5)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.001414	0.021323	0.001625	0.002731	0.000073	0.047047	0.070310
assert/CC	0.000228	0.013978	0.000326	0.000609	0.000569	0.035739	0.051316
Unverified Branchpoints	0.794458	0.256661	0.803209	0.946713	0.618528	0.109849	0.053630
Unverified CC	0.974907	0.486173	0.981893	0.952068	0.772281	0.300915	0.189272
test/prod	0.001613	0.017101	0.001451	0.002013	0.000354	0.017244	0.012428

Table C.119: Pearson correlation for CSharp systems between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.918123	-0.087975	-0.081472	0.228073
assert/CC	0.918123	1.000000	-0.054551	-0.049904	0.168054
Unverified Branchpoints	-0.087975	-0.054551	1.000000	0.932552	-0.064424
Unverified CC	-0.081472	-0.049904	0.932552	1.000000	-0.058206
test/prod	0.228073	0.168054	-0.064424	-0.058206	1.000000

Table C.120: Coefficient of determination for CSharp systems between test code metrics (5)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.842950	0.007740	0.006638	0.052017
assert/CC	0.842950	1.000000	0.002976	0.002490	0.028242
Unverified Branchpoints	0.007740	0.002976	1.000000	0.869654	0.004150
Unverified CC	0.006638	0.002490	0.869654	1.000000	0.003388
test/prod	0.052017	0.028242	0.004150	0.003388	1.000000

Table C.121: Pearson correlation for CSharp systems between test code metrics and simple metrics (6)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.078209	0.076775	-0.079628	-0.094641	-0.061712	0.116561	0.163318
assert/CC	-0.047859	0.121586	-0.047657	-0.057727	-0.029607	0.192765	0.254443
Unverified Branchpoints	0.934894	0.430179	0.934066	0.974770	0.819070	0.341483	0.330874
Unverified CC	0.994693	0.597047	0.995997	0.969292	0.924206	0.527092	0.488299
test/prod	-0.069502	0.202370	-0.066867	-0.088491	-0.046684	0.175462	0.111212

Table C.122: Coefficient of determination for CSharp systems between test code metrics and simple metrics (6)

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.006117	0.005894	0.006341	0.008957	0.003808	0.013586	0.026673
assert/CC	0.002290	0.014783	0.002271	0.003332	0.000877	0.037158	0.064741
Unverified Branchpoints	0.874028	0.185054	0.872480	0.950176	0.670876	0.116611	0.109478
Unverified CC	0.989414	0.356465	0.992011	0.939527	0.854157	0.277826	0.238436
test/prod	0.004830	0.040954	0.004471	0.007831	0.002179	0.030787	0.012368

Table C.123: Pearson correlation for CSharp systems between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.918666	-0.129541	-0.101624	0.448888
assert/CC	0.918666	1.000000	-0.109931	-0.077301	0.462855
Unverified Branchpoints	-0.129541	-0.109931	1.000000	0.951598	-0.116694
Unverified CC	-0.101624	-0.077301	0.951598	1.000000	-0.082625
test/prod	0.448888	0.462855	-0.116694	-0.082625	1.000000

Table C.124: Coefficient of determination for CSharp systems between test code metrics (6)

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.843948	0.016781	0.010327	0.201501
assert/CC	0.843948	1.000000	0.012085	0.005975	0.214235
Unverified Branchpoints	0.016781	0.012085	1.000000	0.905538	0.013617
Unverified CC	0.010327	0.005975	0.905538	1.000000	0.006827
test/prod	0.201501	0.214235	0.013617	0.006827	1.000000

Table C.125: Average Pearson correlations for CSharp systems between test code metrics.

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	-0.063932	0.082163	-0.066110	-0.072335	-0.035212	0.150473	0.229716
assert/CC	-0.037227	0.092974	-0.039413	-0.043030	-0.007557	0.175521	0.257737
test/prod	-0.044797	0.139156	-0.043838	-0.053871	-0.024576	0.149951	0.118242
Unverified Branchpoints	0.921050	0.560705	0.928800	0.982710	0.824825	0.389745	0.229484
Unverified CC	0.991852	0.703623	0.995249	0.973940	0.909265	0.568214	0.393969

Table C.126: Average Coefficient of determination for CSharp systems between test code metrics.

	SLOC_prod	SLOC_test	CC_prod	Branchpoints_prod	asserts_prod	asserts_test	Direct asserts
assert/branch	0.004230	0.008941	0.004502	0.005372	0.001820	0.024760	0.053989
assert/CC	0.001507	0.011732	0.001656	0.001962	0.000707	0.034600	0.069945
test/prod	0.002594	0.023254	0.002481	0.003368	0.001426	0.027944	0.017616
Unverified Branchpoints	0.848932	0.330900	0.863245	0.965788	0.682964	0.161299	0.055263
Unverified CC	0.983778	0.501691	0.990526	0.948595	0.827389	0.327369	0.160020

Table C.127: Average Pearson correlations for CSharp systems between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.920155	-0.099685	-0.092061	0.294548
assert/CC	0.920155	1.000000	-0.074887	-0.065660	0.261340
test/prod	0.294548	0.261340	-0.071335	-0.058238	1.000000
Unverified Branchpoints	-0.099685	-0.074887	1.000000	0.949671	-0.071335
Unverified CC	-0.092061	-0.065660	0.949671	1.000000	-0.058238

Table C.128: Average Coefficient of determination for CSharp systems between test code metrics and simple metrics.

	assert/branch	assert/CC	Unverified Branchpoints	Unverified CC	test/prod
assert/branch	1.000000	0.846844	0.010214	0.008539	0.099898
assert/CC	0.846844	1.000000	0.006090	0.004446	0.081247
test/prod	0.099898	0.081247	0.005671	0.003862	1.000000
Unverified Branchpoints	0.010214	0.006090	1.000000	0.902114	0.005671
Unverified CC	0.008539	0.004446	0.902114	1.000000	0.003862