A Report on Pseudo-Tested Methods

1. Introduction:

Software testing is a method of ensuring if the system meets the desired requirements. The main goal of software testing is to find gaps and errors in design and implementation of the System under test SUT.

Now a days, most commonly used software development life cycle is an iterative and incremental process where the system goes through many changes against each iteration and even after deployment of the software this process continues. During this time the system also need to be tested iteratively for the changes made into it. For this purpose, regression testing techniques are used and evaluated. In Practice code coverage criteria is used to evaluate the effectiveness of any test suite.

The code coverage focus on which parts of the code has executed during the test execution, ignoring the quality of assertions used. Which arises the question of whether code coverage is good criteria for effectiveness of the test suite used for regression testing or not. Niedermayr et al., [1] in their study focus on this and found that in some cases the test coverage was 100% even after the introduction of extreme mutants into the classes (Extreme mutants were created by removing the code logic from the methods) this proves that the test suite providing 100% coverage failed to detect the extreme mutation and hence the methods which passed such mutation test were named as pseudo-tested methods.

Vera-P´erez et al., [2], [3] further worked on the idea of extreme mutation and confirm the existence of pseudo-tested methods in 21 open-source java projects. Additionally, they performed a quantitative analysis which shows how are pseudo-method different from other non-pseudo methods. To ensure the importance of their analysis they performed manual qualitative analysis on 101 pseudo-tested methods and found that 30% of these methods need an additional testing focus. This paper has been used in this report as a base paper.

The focused of this report is to re-create the experiment performed by Vera-P´erez et al., [2] to show the existence of pseudo-tested methods in 21 open-source java projects. I will first discuss the targeted software and issues I have faced. In the next section I have provided my analysis of current results with results generated by Vera-P´erez et al.,[2]. In last I will conclude my report.

2. Targeted Software's:

The number of opensource software's focused in the base paper is given in Table 1. Along with the build status column where "True" shows the status of software which were successfully build and its results were generated. While "False" indicates that there were issues while building the subjects. These issues are discussed in detail here.

2.1 Amazon Web Services SDK:

This project is one of the largest projects among all the targeted project. After investing a lot of time trying to build this project with different versions of JDK and maven, I failed to do so due to failure of several test cases in the existing test suite. After careful investigation I came

across that may be it is because of using windows 10. This issue is still open on GitHub with the following link https://github.com/aws/aws-sdk-java/issues/1981 .

2.2 Jaxen XPath Engine:

After trying different version of JDK and maven combination with this project the project was built successfully along with all test cases passed but PItest fail to detect any test with error "*TestEngine with ID 'junit-jupiter' failed to discover tests* ".

2.3 Java Git:

The project is no more available on GitHub following the link given in base paper.

2.4 Joda-Time:

The project also faces the same problem as Jaxen Xpath Engine as the PItest fails to detect and tests. Another issue is that Joda-time is no longer in active development except to keep timezone data up to date. Java.Time is now alternatively used which has been added to Java SE 8 and onwards.

Table 1: List of software's targeted in base paper

S.no	Project Name	Project ID	Build Status
1	AuthZForce PDP Core	authzforce	True
2	Amazon Web Services SDK	aws-sdk-java	False
3	Apache Commons CLI	commons-cli	True
4	Apache Commons Codec	commons-codec	True
5	Apache Commons Collections	commons-collections	True
6	Apache Commons IO	commons-io	True
7	Apache Commons Lang	commons-lang	True
8	Apache Flink	flink-core	True
9	Google Gson	gson	True
10	Jaxen XPath Engine	jaxen	False
11	JFreeChart	jfreechart	True
12	Java Git	jgit	False
13	Joda-Time	joda-time	False
14	JOpt Simple	jopt-simple	True
15	jsoup	jsoup	True
16	SAT4J Core	sat4j-core	False
17	Apache PdfBox	pdfbox	False
18	SCIFIO	scifio	False
19	Spoon	spoon	False
20	Urban Airship Client Library	urbanairship	True
21	XWiki Rendering Engine	xwiki-rendering	True

2.5 SAT4J Core:

While building this project many of the test cases failed. Due to which it is not possible to apply Descartes as it need a green test suite.

2.6 Apache PdfBox:

The test suite provided is not green and many of the test failed to execute. Hence Descartes cannot be applied to it

2.7 SCIFIO:

This project has more than 12K commits on its GitHub Repo. PI test fails to detect any test cases in this project. Understating of this behavior need further investigation.

2.8 Spoon:

The project was unable to be build along with all test cases. The issue has been raised on GitHub. Issue link https://github.com/INRIA/spoon/issues/4745.

3. Analysis:

After cleanly building the remaining projects. I applied the newer version Descartes (1.3.6) which is an open-source project develop by the authors of base paper. The latest version excludes all the stop methods by default while in older version it was not the case and the stop methods were removed after the final results were generated by Descartes. Stop method normally includes methods shown in Table 2

Table 2: Stop Methods

Methods	Method description
empty	void methods with no instruction.
enum	Methods generated by the compiler to support enum types (values and valueOf).
to_string	toString methods.
hash_code	hashCode methods.
deprecated	Methods annotated with @Deprecated or belonging to a class with the same annotation.
synthetic	Methods generated by the compiler.
getter	Simple getters.
setter	Simple setters. Includes also fluent simple setters.
constant	Methods returning a literal constant.
delegate	Methods implementing simple delegation.
clinit	Static class initializers.
return_this	Methods that only return this.
return_param	Methods that only return the value of a real parameter
kotlin_setter	Setters generated for data classes in Kotlin (*New in version 2.1.6*)

To generate mutants Descartes, apply extreme mutation i.e., the body of the method is striped out generating new variants by simply returning predefined values depending on the return type of the methods under analysis. The details of this extreme transformation are given in Table 3.

Furthermore, the new version is capable of classifying the methods under analysis into four different categories i.e. *(tested, partially-tested, pseudo-tested, not-covered)*. A method is said to be not-covered if it is ***not covered*** by the test suite. It is said to be ***pseudo-tested*** if it is

covered by the test suite, yet no extreme mutation applied to the method was detected by any test case. A method is said to be ***partially-tested*** if test has mixed results: some mutations were detected and others were not. Finally, a method is classified as ***tested*** if all extreme transformations are detected by the test suite.

Table 3: Extreme Transformation

Methods Return Type	Values used
void	-
Reference types	null
boolean	true,false
byte,short,int,long	0,1
float,double	0.0,0.1
char	‘ ’, ‘A’
String	“”, “A”
T[]	new T[] {}

The results achieved by applying Descarates (1.3.6) are given in Table 4 along with the results generated previously by the base paper.

In Table 4, **#MUA** represent the number of methods under analysis. **#PSEUDO** represent the number of pseudo-tested methods. **PS_RATE** indicates the pseudo-tested methods ratio among the #MUA, **#PT** represent the number of partially-tested methods and **PT_PS RATE** represent the overall rate of partial and pseudo-tested methods. Here the subscript “**2**” represent the data of experiments performed with newer version while the subscript “**1**” shows the data presented in the base paper.

Table 4:Results of descarates

Project	#MUA-1	#PSEUDO-1	PS_RATE-1	#MUA-2	#PSEUDO-2	#PT-2	PS_RATE-2	PT_PS RATE-2
authzforce	291	13	4%	423	11	8	3%	4%
commons-cli	141	2	1%	137	1	2	1%	2%
commons-codec	426	12	3%	559	9	16	2%	4%
commons-collections	1232	40	3%	2663	74	22	3%	4%
commons-io	641	29	5%	1065	33	12	3%	4%
commons-lang	1889	47	2%	2387	30	45	1%	3%
flink-core	1814	100	6%	3659	84	104	2%	5%
gson	477	10	2%	575	6	11	1%	3%
Jfreechart	3496	476	14%	4704	412	83	9%	11%
jopt-simple	256	2	1%	236	2	1	1%	1%
Jsoup	751	28	4%	1062	6	12	1%	2%
Urbanairship	1989	28	1%	2246	62	106	3%	7%
xwiki-rendering	2049	239	12%	862	5	6	1%	1%

To understand the difference, we compare the results of PS_RATE-1 with PS_RATE-2 in Figure 1.

During our analysis we found that in most of the projects the PS_RATE is either decreased or remains equal, which is good sign that developers are now trying to reduce the PS_RATE and making their test suite stronger. The only outlier here is Urbanairship project where the PS_RATE has been significantly increased.

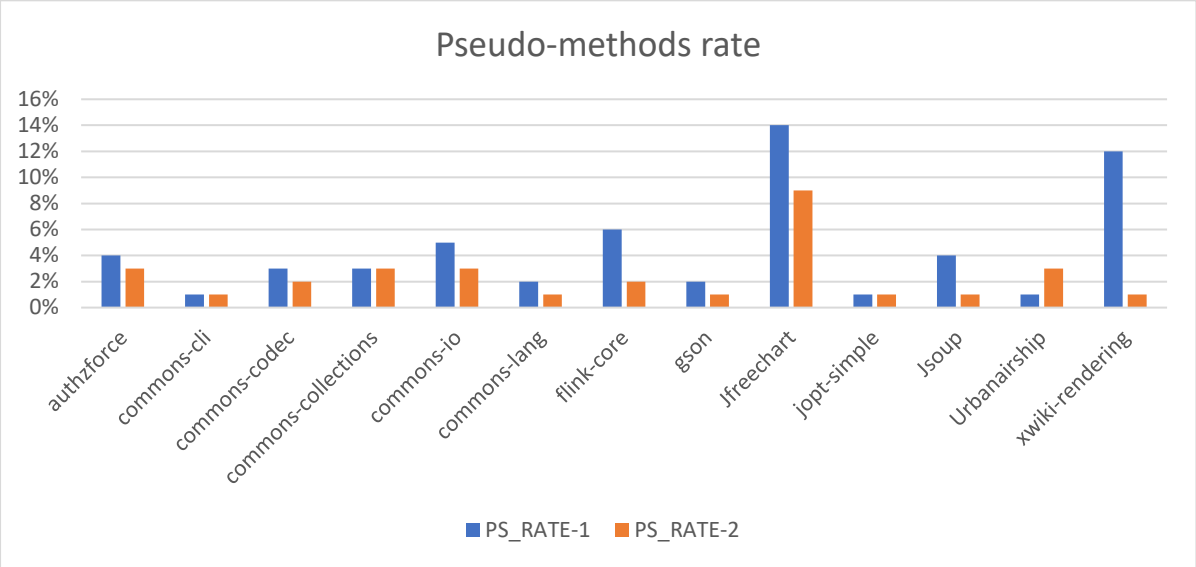

Figure 1: Pseudo-tested methods rate

Figure 2 here show analysis between PS_RATE-1 and PS_PT_RATE. As Partially tested methods are also Pseudo tested so combing the results of both PS and PT can give us further detail insights about the improvements made over the time in different projects. The results show that no significant difference in results for all project except Commons-CLI project where the rate has been increased.

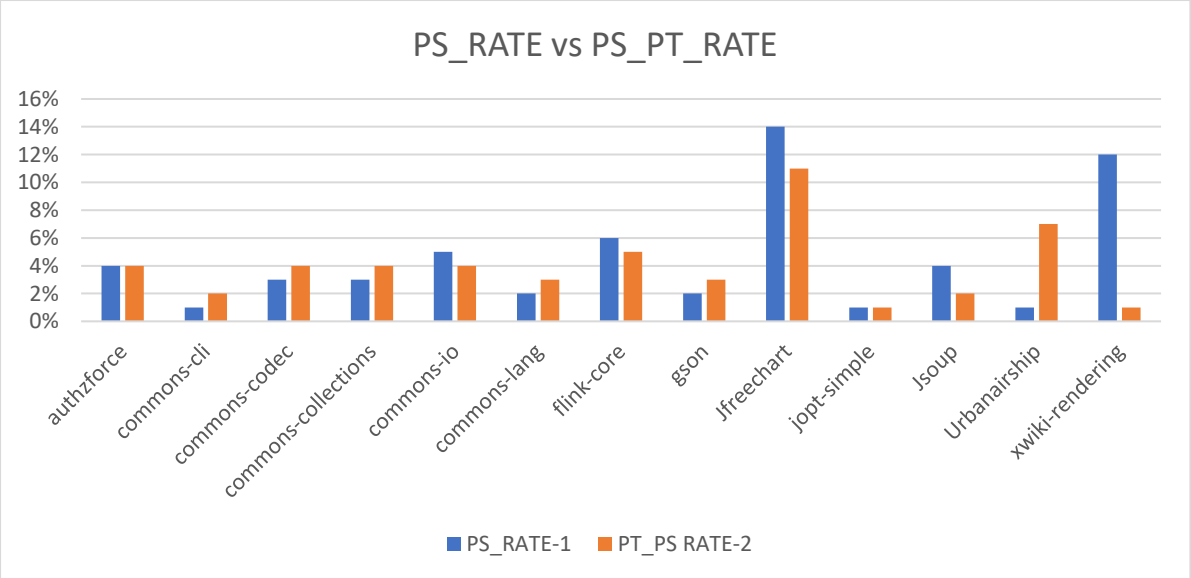

Figure 2: PS_RATE vs PS_PT_RATE

4. Conclusion:

After analyzing all the study subject given in the base paper. I was able to execute many of them and generate the results as required. During this process I faced many difficulties and

issues mostly due to version. In some cases, the test suit provided was able to pass the test suite which indicates errors in the underlying subject, due to which it was not possible to generate their results. Overall, it was quite interesting to know about the pseudo-tested methods and its existence in well-known open-source projects. In last by comparing the results we can clearly deduce that many of the subjects have improved their test suites to tackle the pseudo-tested methods problem.

All the code and data generated by my experiments is available through the following link:

https://github.com/khwbilal/pseudo-tested-methods

5. Reference:

[1] R. Niedermayr, E. Juergens, and S. Wagner, "Will my tests tell me if I break this code?," in *Proceedings - International Workshop on Continuous Software Evolution and Delivery, CSED 2016*, May 2016, pp. 23–29. doi: 10.1145/2896941.2896944.

[2] O. L. Vera-Pérez, B. Danglot, M. Monperrus, and B. Baudry, "A Comprehensive Study of Pseudo-tested Methods," Jul. 2018, doi: 10.1007/s10664-018-9653-2.

[3] O. Luis Vera-Pérez, M. Monperrus, and B. Baudry, "Descartes: a PITest engine to detect pseudo-tested methods-Tool Demonstration Descartes: a PITest engine to detect pseudo-tested methods-Tool Demonstration Descartes: A PITest Engine to Detect Pseudo-Tested Methods Tool Demonstration," pp. 908–911, 2018, doi: 10.1145/3238147.3240474ï.