SOEN 6011- SOFTWARE ENGINEERING PROCESSES
PROJECT DELIVERABLE 3

Himani Patel (40071101)
Concordia University, Montreal

GitHub Link : `https://github.com/Himani-R-Patel/SOEN-6011`

August 02 2019

Contents

0.1 Problem 5

0.1.1 Code Review

0.1.1.1 Introduction

Code review also referred to as peer review is a software quality assurance activity in which one or several humans check a program mainly by viewing and reading parts of its source code and giving the feedback, and they do so after implementation or as an interruption of implementation.The main goal of the code review is the direct discovery of quality problems. Code review is the most commonly used procedure for validating the design and implementation of features. It helps developers to maintain consistency between design and implementation styles across many team members and between various projects on which the company is working.

0.1.1.2 Assigned Function to review : Function1 : Arccos(x)

Arccos(x) generally define as the inverse of the trigonometric Cosine function.The arc-prefix depicts the inverse part of the function.Domain of the function. The domain of the function is restricted to set of numbers between -1 to 1 both inclusive. Due to the mapping in case of Arccos function being many-to-one inverse function would not satisfy any one range. Where as the arccos function defines in the range of 0 to π.

This report is briefly describe the summary of the code review conducted on the aforementioned function.

0.1.2 Code Review Approach

In general, manual code review can become tedious and error-prone,especially as the size of the source code grows. This can be circumvented somewhat by automation by an appropriate use of tools. However, the tools also have limit on capabilities such as meaningfulness of names, quality of comments, or choice of algorithms.For such a cases, a tool may not be useful. So that, the code review process cannot be automated entirely. To conclude, the combination of the manual and automatic code review process would be the better option rather than to select either of them.

Hence, the code review of the given function is also performed manually as well as automatically by using the PMD(a source code analyzer and a copy-paste-detector) plugin for the eclipse. The reason to select this tool is that it provides a complete violations overview and outline of the reviewed code(Figure 2 and Figure 3 respectively).It also provides the detail description of the individual violation with underlying rule name(Figure 4). It also allow the reviewer to remove the violation , mark as reviewed and more importantly disable the underlying rule. Indeed , the customized code review can be performed.

0.1.2.1 Automated Source Code Review

In order to perform source code review automatically, the PMD source code analyzer was used as a plug-in for eclipse IDE. Both classes of the project has been reviewed individually. In PMD , all the violations are categorized in to five different categories which can be seen as different colours namely red stands for blocker violations, blue stands for critical violations, green color stands for urgent violations, pink are important violations and navy blue represents warnings. They are in order of severity of the violations from highest to lowest which means blocker violations is more important to remove from the source code compared to the others.

After reviewing the class with the tool it gives 88 violations in total from both of the classes. However, most of the violations are related to high level programming style or type of which has not been considered as important for this project.The example of such violations are Data Flow Anomaly Analysis , Method Arguments Could Be Final , Local Variable Could Be Final, Comment Default Access Modifier, Pre-mature Declaration and so on. Therefore, the rules underlying those violations has been disabled to perform the code review. As a result, the total number of violations has reduced from 88 in total from both classes(as can see in Figure 1) to 42 in total from both classes(as can see in Figure 2).

As you can see in the following figures, some of the common violations are Use of Short Variables(Figure 5), Use of Confusing Ternary(Figure 6), Avoid Literals in the If Condition (Figure 7), No package(Figure 8), Use of default package , Class Naming Conventions and so on.Each violation gives the description of why it has been considered as violation(reason), the particular line number at which the violation has occurred So it is easy for developer to resolve such a violation and improve the quality of the source code.

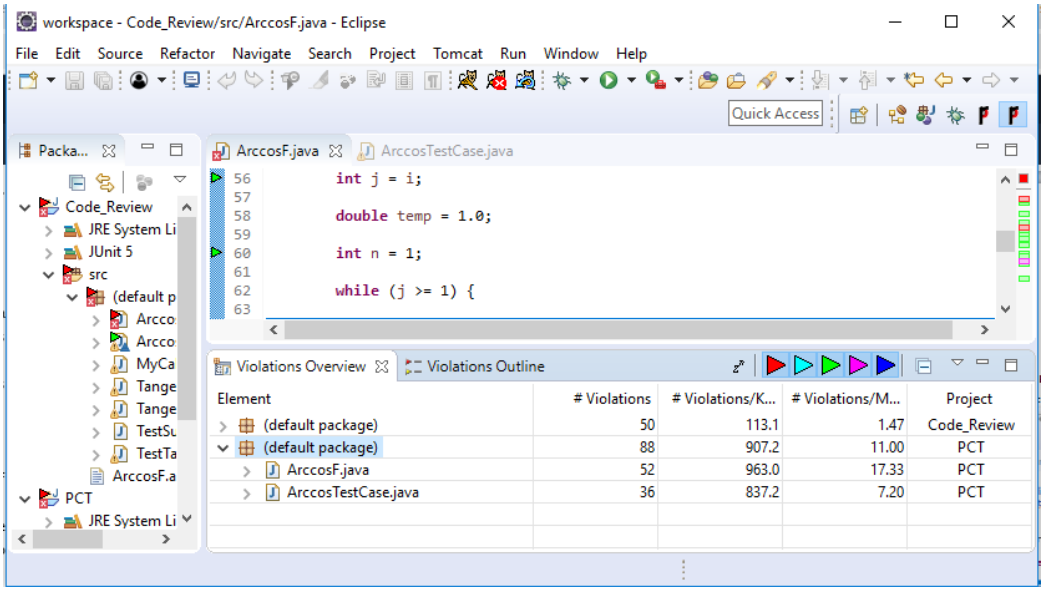

Figure 1: Total number of violations before disabling the rules

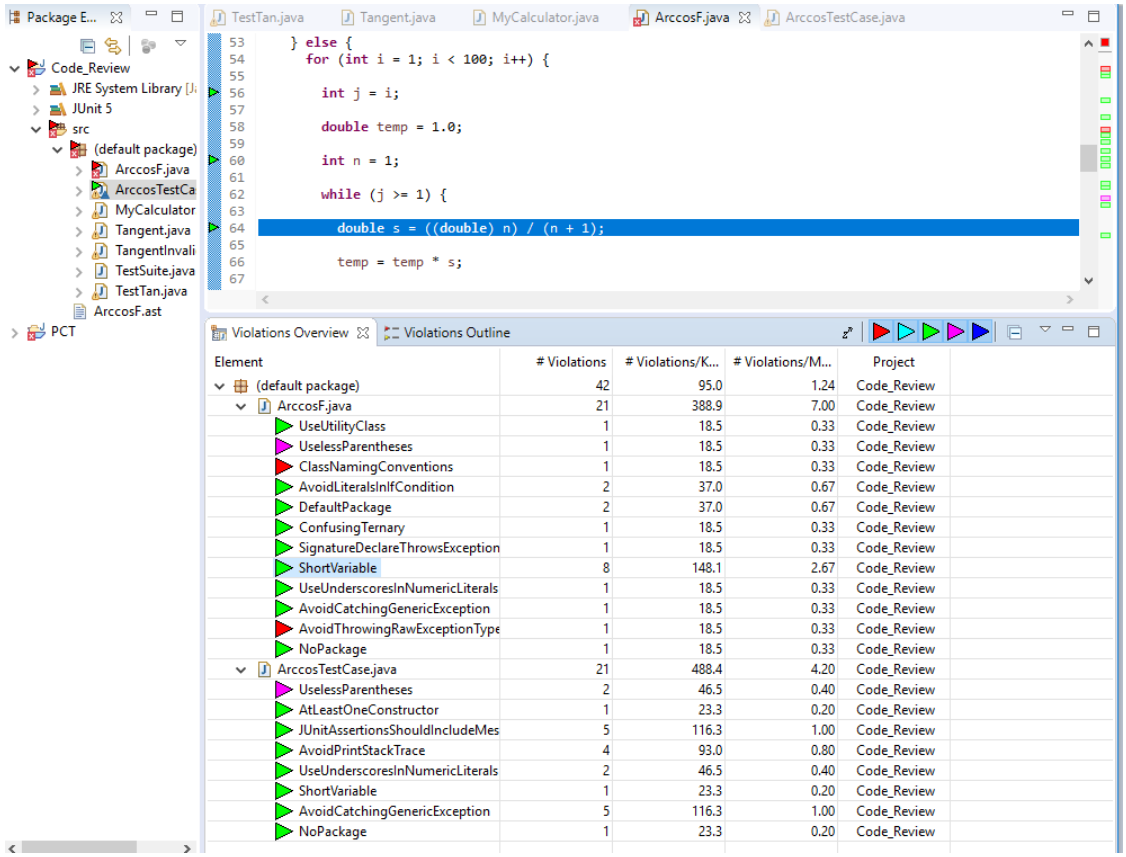

Figure 2: Total number of violations after disabling the rules and Violation overview

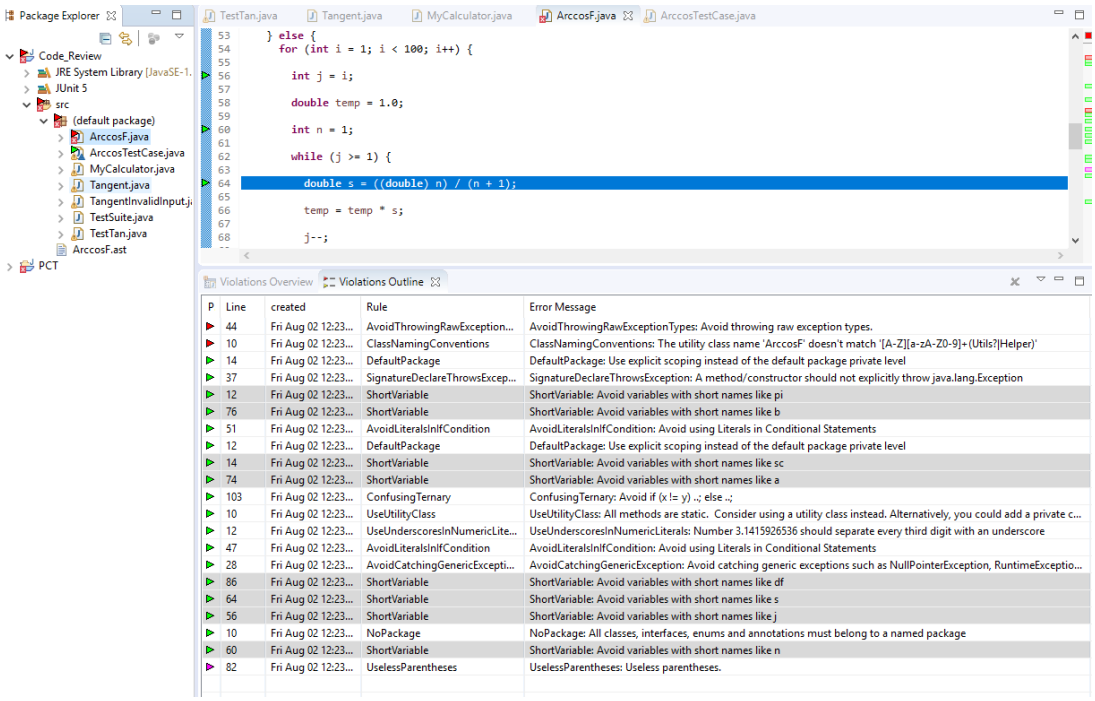

Figure 3: Violation summary of the ArcossF class

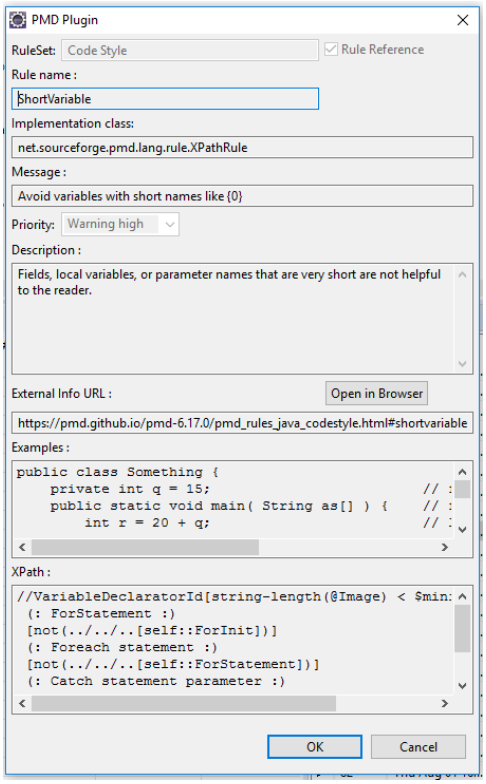

Figure 4: Detail description of the ShortVariable Violation

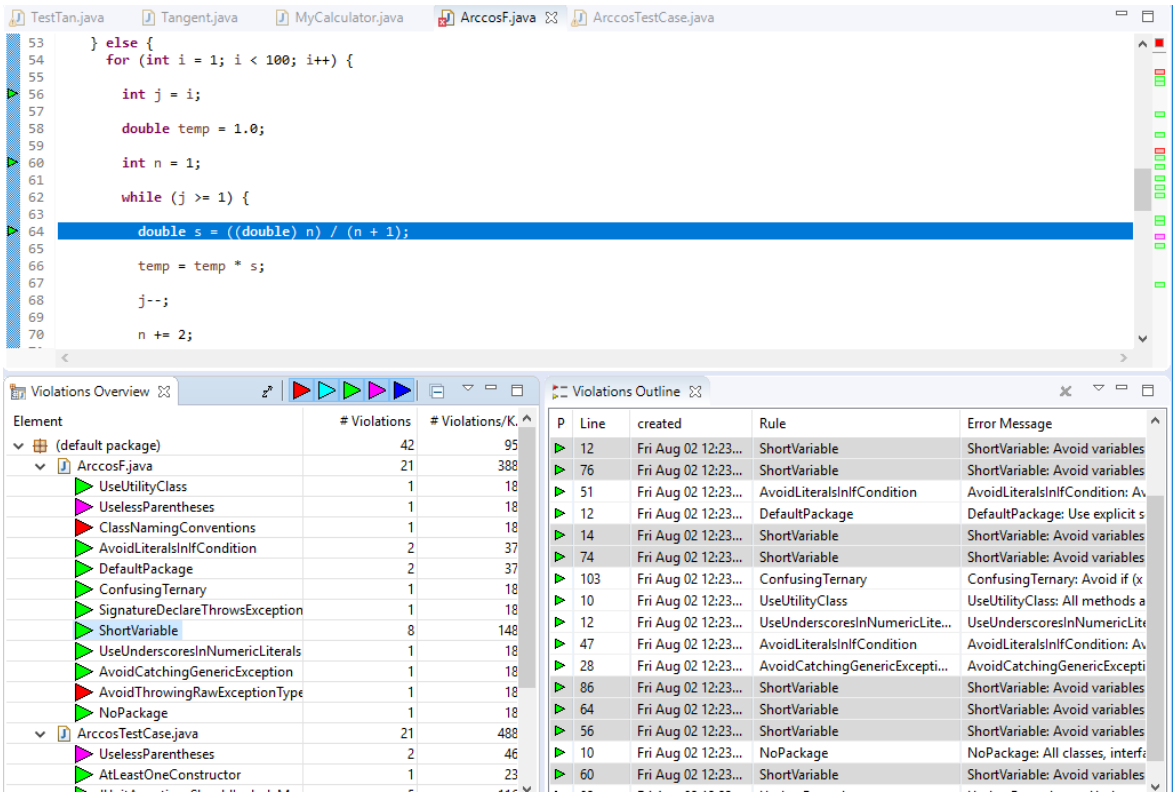

Figure 5: Violation of the Short Variable for ArcossF class

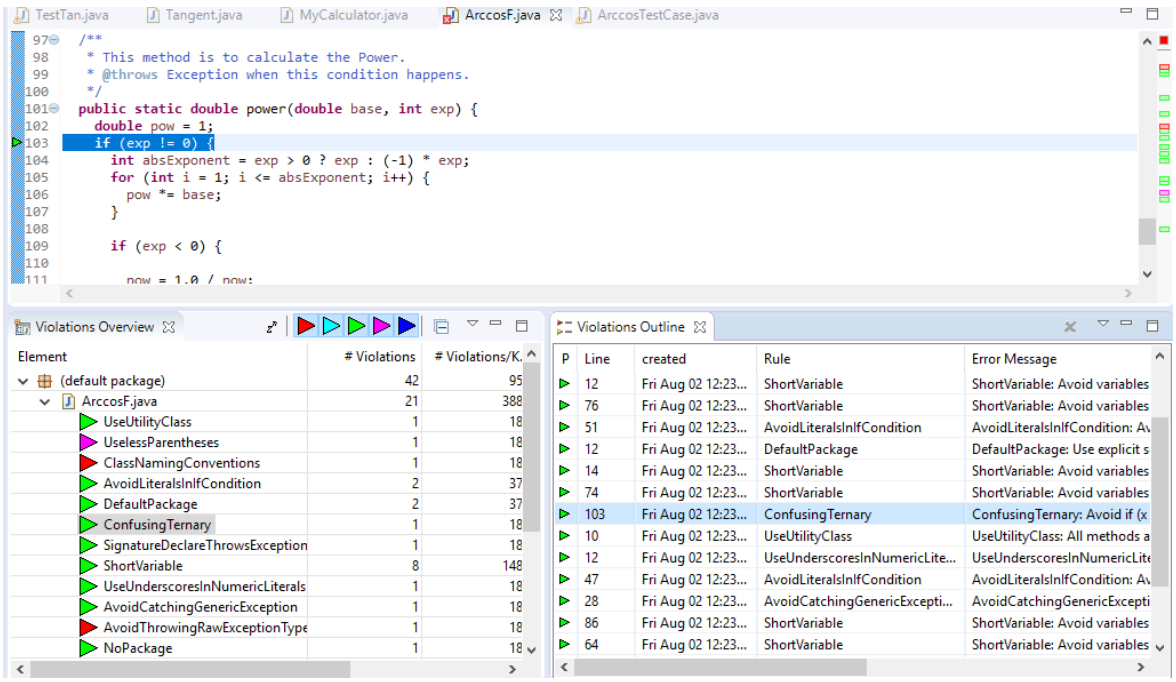

Figure 6: Violation of the Confusing Ternary for ArcossF class

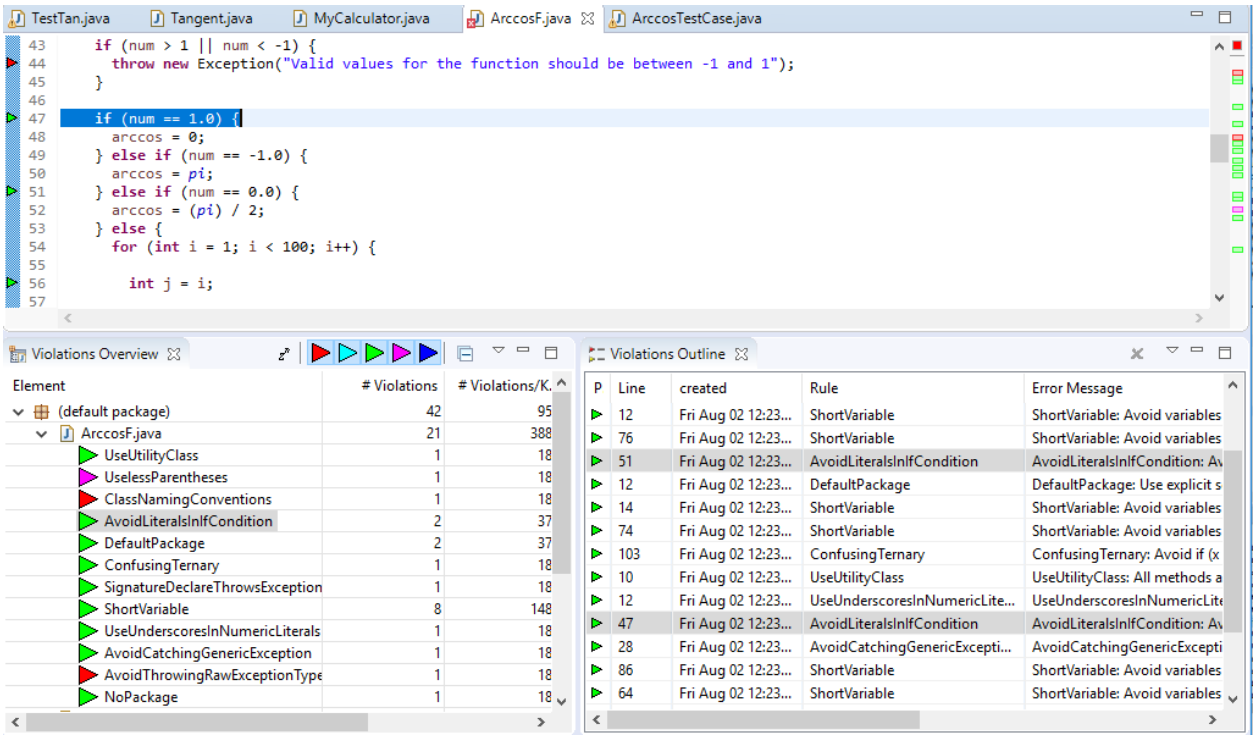

Figure 7: Violation of the Avoid Literals in If for ArcossF class

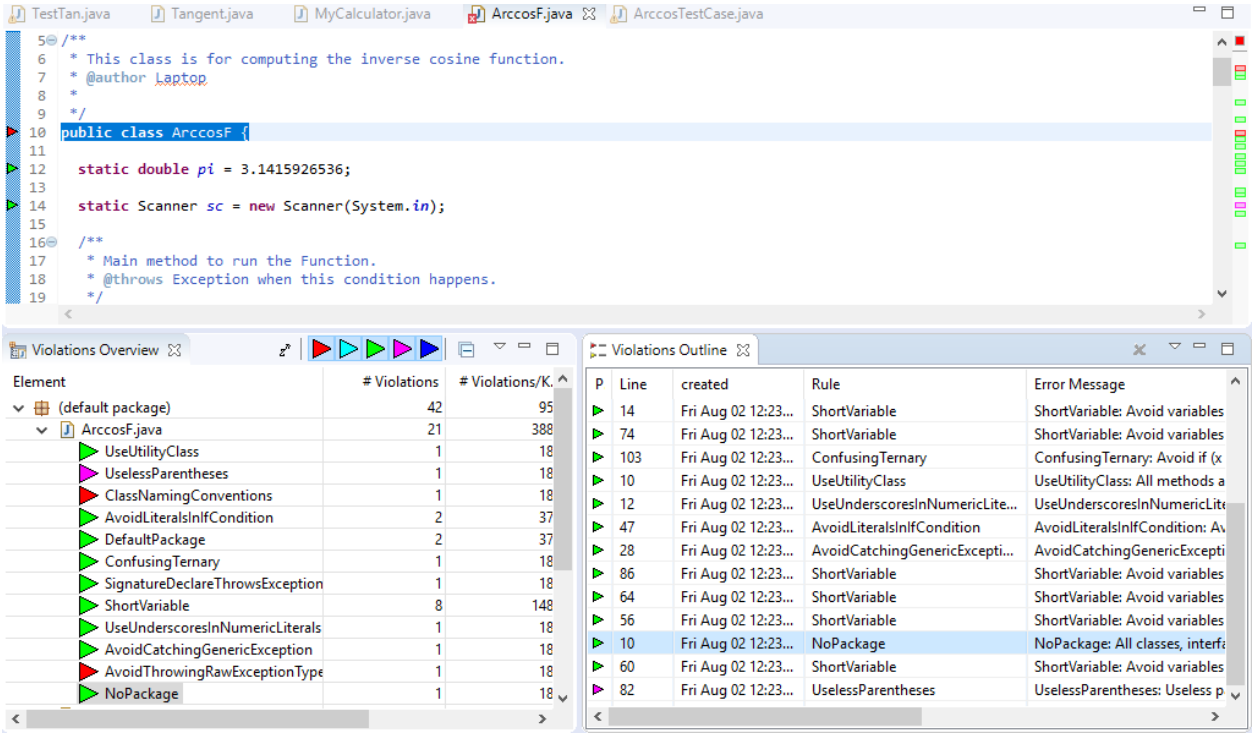

Figure 8: Violation of the No Package for ArcossF class

0.1.2.2 Manual Source Code Review Approach

The manual source code review has been performed by considering the various aspects as follows:

- **Meaningful comments :** Meaningful comments can reduce consumption of time in reading and understanding source code.Following aspects has been reviewed in order to check for the meaningful comments :
  - Source code should maintain consistency between source code and comments.
  - Use of comment at the beginning of class with meta information such as description and author.
- **Naming conventions :** Variable names, method names, class names should follow the Camel case.
- **Meaningful name :** It has been shown in the studies that appropriate naming is significant for economically- and technically-effective evolution of Java programs. Naming conventions has been considered here are as follows:
  - **Constant names:** It should be in all capital letters and separate internal words with the underscore character.
  - **variable names :** Permanent variables should have longer and more descriptive names whereas temporary variables should have shorter names.
  - **Class name :** It should be noun and begin with the Capital letter.
  - **Method name :** It should be verb and begin with the lower case letter.
- **White space :** It can enhance readability of the source code. Indeed , it also enhance the understandability as well as for maintainability of the source code.Following aspects has been reviewed in order to check the proper use of white spaces :
  - Use of blank lines to separate source code into logical sections.
  - Use of single space character between all binary operators.
  - Use of single space character after each statement in a for loop, after each comma in an argument list and after each comment delimiter.
- **Indentation :** It can enhance understanding of the source code and searching the source code manually, say, for debugging. Following aspects has been reviewed in order to check the proper use of indentation.

  - Avoid lines longer than 100 characters.
  - Indented a fixed number of spaces
  - Use of new indentation level for each level of nesting in the source code.

- **No long code :** Code should fit in the standard computer screen. So that, there should not be a need to scroll horizontally to view the code.

0.1.2.3 Manual Source Code Review Summary

- **Meaningful comments :**
  - All comments have been used are meaningful and understandable. However, none of the comments for the method have description about the **parameters.**
  - The comment at the beginning of class with meta information have been used in ArcossF class but it has not been used in the respective test class. However, the name of author in the given comment is **Laptop.** It is better to use real author name.

- **Naming conventions :** Have used camel case properly. The naming style should be consistent in the source code but here in some places the **Arccos** have used whereas in other places **Arcos** have used.

- **Meaningful name :**
  - Name of the constant PI is in **lower case.** It should be in capital letters.
  - Variable names such as **rad , deg** have been used are not descriptive. It should be used instead **radian , degree** respectively. In addition , the variable names such as **temp, s, b** have been used have no meaning. It is better to use some **relative name** .
  - Class name have been used correctly. However, it is also better to use Arcoss-Function instead of **ArcossF.**
  - All methods name have been used correctly except **power method.** The name of the method should be verb instead of noun. So that, it is better to use CalculatePower instead of the power.

- **White space :**

  - Have properly used blank lines to separate source code into logical sections.
  - Have properly used single space character between all binary operators.
  - Have properly used single space character after each statement in a for loop, after each comma in an argument list and after each comment delimiter.

- **Indentation :**

  - All lines of code have been used are less than 100 characters.
  - Source code have been properly indented with a fixed number of spaces.
  - New indentation level have been used properly for each level of nesting in the source code.

- **No long code :** Code has fitted in the standard computer screen.

- **Other:**

  - The Source has compiled successfully.
  - Exception handling has done properly.

0.2 Problem 7

0.2.1 Test Case Analysis

A TEST CASE is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.The process of developing test cases can also help find problems in the requirements or design of an application.

0.2.1.1 Assigned Function to test : Function2 : tan(x)

In any right triangle, the tangent of an angle is the length of the opposite side (O) divided by the length of the adjacent side (A). In a formula, it is written simply as 'tan'
$tan(X) = \frac{o}{A}$

This report provides a brief summary of the test cases and their result which has been created for the Tangent function.

0.2.2 Test Case Analysis Summary

In the TestTan class , the author has created 7 test cases in order to test the method which has been implemented. Out of which the last test case namely Test1 has no meaning since it is not performing any comparison of the results with the implemented function means it has not been implemented to test any of the method and it is just comparing the two same values. In addition , there is no test case which tests fac method (which validates the calculation of the factorial of the given number) of the Tangent class.

By analyzing the test cases and the functional requirements, it has been found that the written test cases not validates the given either of two functional requirements. However, the given both functional requirements can be validated by entering the value which has given in the functional requirement through User Interface of the system. To reciprocate, the functional requirements and the test cases can be tested individually but they are not related with each other.

The written test cases are as follows : testTan1() and testTan2() tests the tan(which calculates the Tangent) method of the Tangent class, testCos(),testCos2() tests the cos(which calculates the cosine) method of the Tangent class, testSin() tests the sin(which calculates

the sine) method of the Tangent class, powerTest() tests the CalculatePower method of the Tangent class.

The given figure shows that all of the test cases have successfully passed.

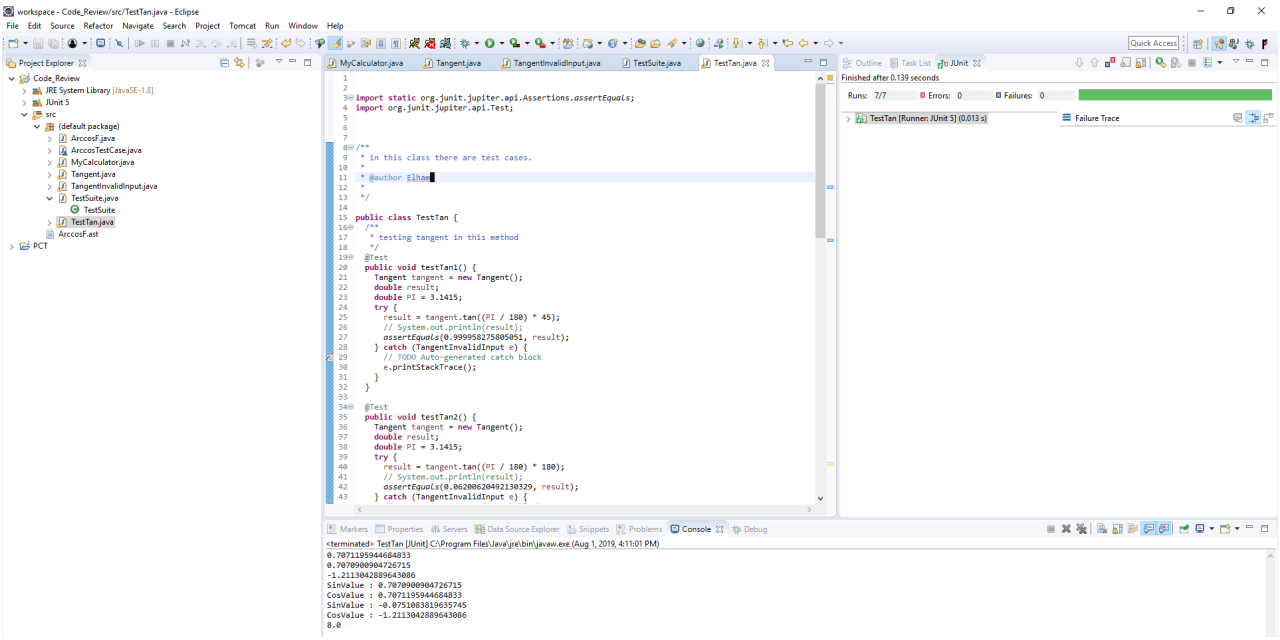

Figure 9: Progress Report of JUnit test cases

0.2.3 Test Case Analysis Summary Tables

The following table shows the summary of the generated output of the test cases on giving the input that has provided by the author of the function application. In addition, the summary table includes the expected output as well as the output generated by the particular method in the Tangent class along with the status of the test case which can be either Passed or Failed. It can be shown from the table that , all the test cases have passed successfully which means the application is working perfectly for the author's given input.

Test Case For Method: tan()

Input	Expected value	Obtained value	Status
45	0.999958275805051	0.999958275805051	Passed
180	0.06200620492130329	0.062006204921303	Passed

Test Case For Method: sin()

Input	Expected value	Obtained value	Status
45	0.7070900904726715	0.7070900904726715	Passed

Test Case For Method: cos()

Input	Expected value	Obtained value	Status
45	0.7071195944684833	0.7071195944684833	Passed
180	-1.2113042889643086	-1.2113042889643086	Passed

Test Case For Method: powerTest()

Input	Expected value	Obtained value	Status
2, 3	8	8.0	Passed

Bibliography

[1] Wikipedia
https://en.wikipedia.org/wiki/Code_review

[2] EvokeTechnologies
https://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-cod

[3] Java Coding Style Guide
https://www.cis.gvsu.edu/java-coding-style-guide/

[4] Thinkapps
http://thinkapps.com/blog/development/what-is-code-review/

[5] Introduction to Java Programming style [kamthan,2018]
https://users.encs.concordia.ca/~kamthan/courses/soen-6011/
programming_style_java_introduction.pdf

[6] Software Testing Fundamentals
http://softwaretestingfundamentals.com/test-case/