Assignment #1 Software Testing

Com S/SE 417 Spring 2024

Handed out Jan 25th 2024

Due at 9 PM, Feb 6th, as a pdf uploaded to Canvas

Homework Policy

Homework Policy: The homework assignment should be done individually. You may talk to classmates about the problems in general, and you can help each other with getting the tools running, but you must complete the homework on your own. You are not permitted to use published answers from websites, etc. Assistance by others must be specifically credited in the solution to the problem that is turned in, describing what the contribution was (e.g., "Thanks to [name] for explaining X in Problem Y", not "Thanks to [name] for help with HW1."). The Dean of Students Office offers several good resources on how to avoid plagiarism.

The goal is for each of you to learn the material sufficiently well to use it productively, to think innovatively, and to develop confidence in your problem-solving abilities. Feel free to talk to me individually about this if you have any questions.

Late policy: *10% penalty per day* (or part of day) for late homework. ***Assignments will not be accepted after Feb 9th*, 9PM** unless otherwise arranged/discussed with me.

Late Policy One Slack Period for all Homework

During the semester you have one "slack period". This means that you can hand in one homework late – up until the late acceptance date (e.g. Feb 9th for this assignment) without penalty. No questions asked. Once you use up your slack period late penalties will apply as per the late policy

Some homework problems are adapted from the course textbook, "Introduction to Software Testing", 2nd edition, Ammann & Offutt, 2017. There is a Student Solution Manuals available online with answers to other practice questions https://cs.gmu.edu/~offutt/softwaretest/exer-student.pdf.

In this assignment we will use a code coverage tool for Java, Jacoco (version 0.8.11)

https://www.eclemma.org/jacoco/

I have put a version in Canvas with the triangle program and demoed this in class as well as used for our in-class exercise. Please see directions for running Jacoco with that Program. You can use that infrastructure to start this assignment. All you will need to do is to add a new directory inside of the examples folder with the files for this assignment.

I have put a zipped file of a directory called "**prime.tar.gz**" containing a java class PrimeNumberFinder.java that computes a list of prime numbers between an upper and lower bound. These are inclusive (e.g. 2,5 would include primes 2,3, and 5). You need to copy prime.tar.gz to the

"examples" directory in Jacoco and extract it in that location. It will make a directory called prime with the necessary jacoco pom.xml file.. Jacoco will run out of the box on pyrite. You can also run this locally on your own machines (if you want to set it up). You will need Maven and Java to run this on your home machine. I have provided directions for running on Pyrite if you choose to use that version.

Program Specifications: PrimeNumberFinder.java

A prime number is a whole number greater than 1 whose only factors are 1 and itself. The first five primes are 2, 3, 5, 7, 11. It should be noted that 1 is a non-prime number.

This class has three methods The first called findPrimes(int lower, int upper) returns a list of prime numbers between (inclusive) the lower and upper bound. For instance findPrimes(3,6) should return 3,5. The second method called isPrime(int num) returns a Boolean indicating if num is a prime number. For instance, isPrime(3) would return true and isPrime(4) would return false. The last method is called computeSumOfPrimes(list) takes a list of numbers and returns their sum. Although it is in the same class and can be used to compute a sum, it could be used on any list of numbers. *There is a main method that is commented out (Please leave this commented out during your experiments). You are welcome to uncomment the main and try compiling/running the program on the command-line to learn more about how the program works.*

Inside the src/tests/ directory there is a class with some existing Junit tests which you can use to start. You don't need to create new types of tests but will want to add to the test cases (you can copy/paste and change the names and values of existing ones). The important part will be to select the necessary data and oracles to put into the tests.

For each numbered item below complete the actions and answer the questions by providing written answers and giving screen shots demonstrating the resulting reports from Jacoco. This should include the Junit test details, the coverage report and the visualization of the source code showing coverage. Create a .pdf of the final report and submit via canvas. Make sure to clearly label the question #s.

1. Download the Jacoco distribution from their website or from Canvas. Download the 'prime' directory from canvas and place the contents inside of the examples directory as well. If you use a new installation of Jacoco you will need to create the examples directory yourself. (nothing to hand in)

2. Set up the Jacoco program to run either on pyrite and/or your local machine. Run the initial test cases using the "mvn" commands as shown in class (and provided on the lecture notes). This is run inside of the **prime** main directory
   i.e. directory … jacoco-0.8.11/examples/prime (nothing to handin)

3. Look at the initial reports (in the target directory) for JUnit (under surefire-reports) and for Code Coverage (under sites). You should keep these (to hand in). Remember, every time you run mvn clean this directory gets deleted.

Write a short (sentence or two) summary of what these reports tell you and attach the reports to the assignment. Make sure to include the report which shows the code of the class as well.

4. Start by trying to cover the 'isPrime' function. Add additional tests to increase both the line and branch coverage as high as you can. Aim for 100 percent coverage **of this method**, however this may not be possible – **do not change the program** to achieve higher coverage. You must do this **only with test cases**.

(a) **List (show) the test cases you have added and for each state** The input, the oracle, and the reason you added it (i.e. what condition/statement are you trying to cover – give line numbers from the Jacoco report) You can provide the 'actual JUnit code or just the inputs/oracles for each) (b) **Provide screen shots of the reports demonstrating the new resulting coverage. Make sure to show the code screenshot so we can see what has/has not been covered.** (c) What is the **maximum line and branch coverage** the method you have reached? Discuss challenges you faced in reaching this coverage and why you think you were not able to reach 100 percent (if it was not possible).

5. Now add test cases to cover the **other methods**. Again, aim for 100 percent coverage – this time of the entire program.

(a) Show the final coverage you have obtained. (b) Discuss some of the tradeoffs for covering code by the various methods in the class (i.e. are some easier/harder – do you think this impacts the final quality of the test cases). (c) For the computeSumOfPrimes method it really has nothing to do with prime numbers. It will return a sum even if the numbers are prime. Comment on whether this is a good design or if the specifications for that method should require it to perform some data checking or force it to be used only with the findPrimes method.

6. Identify at least two **faults *in the program and one 'exception'*.** These may or may not be in the same functions. You may need to look at both the code and the surefire reports to identify what is happening.

For each, provide a test case that found the fault.

7. **Try fixing the faults you have found** and confirm that the faults are no longer found.

Hand in the changed code and the new Junit report. Do not change the test cases. If you were unable to reach 100 percent code coverage see if you can make changes to fix that as well (without making any of the tests fail) and explain why that code can be changed without losing any program correctness. They should all pass now.