HW #1 Software Testing

Com S/SE 417 Fall, 2018

Due at beginning of class Tuesday, Sept. 18 as pdf uploaded to Canvas

Textbook reading assignment: Chapters 1-5, Amman & Offutt, 2nd ed.

Homework Policy

Homework Policy: The homework assignment can be done individually, or collaboratively by a team of two to four students. Collaboration is permitted if and only if the work is done *jointly and together (physically co-located*), not divided up with each person doing a part of it. If done collaboratively, one solution paper is turned in with all the names on it.

Solutions should be original and independently developed by the individual or small team. No consultation of other people's solutions (published or unpublished) is permitted. No sharing of papers or files with solutions is permitted. 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 to build understanding of how to avoid plagiarism, such as Purdue's "Safe Practices" site.

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 school day for late homework. For example, if an assignment due Tuesday is received by me Tuesday after class, or delivered to my office while I'm in class, it is one day late; if received Wed. after 9:30 it is two days late; if received the following Monday after 9:30 it is five days late, etc. Please contact me in case of emergencies.

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 similar questions https://cs.gmu.edu/~offutt/softwaretest/exer-student.pdf. The JUnit review slides are posted on Canvas.

1. [Chap. 1, Problem #5 in 2nd ed.] Below is a faulty program and a test case that results in failure. Answer the following questions about the program.

```
/**
 * Find last index of element
 *
 * @param x array to search
 * @param y value to look for
 * @return last index of y in x; -1 if absent
 * @throws NullPointerException if x is null
 */
public int findLast (int[] x, int y)
{
   for (int i=x.length-1; i > 0; i--)
   {
      if (x[i] == y)
      {
         return i;
      }
   }
   return -1;
}
// test:  x = [2, 3, 5]; y = 2; Expected = 0
// Book website: FindLast.java
// Book website: FindLastTest.java
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

(b) If possible, give a test case that does not execute the fault. If not, briefly explain why not.

(c) If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.

(d) If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don't forget about the program counter.

(e) For the given test case, describe the first error state. Be sure to describe the complete state.

(f) Implement your repair and verify using JUnit that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

2. [Chap. 1, Problem #7 in 2nd ed.] Consider the following example class. The OO faults are taken from Joshua Bloch's Effective Java, Second Edition. Answer the following questions about it.

```
class Vehicle implements Cloneable {
   private int x;
   public Vehicle(int y) { x = y;}
   public Object clone() {
      Object result = new Vehicle(this.x);
      // Location "A"
      return result;
   }
   // other methods omitted
}
class Truck extends Vehicle {
   private int y;
   public Truck(int z) { super(z); y = z;}
   public Object clone() {
      Object result = super.clone();
      // Location "B"
      ((Truck) result).y = this.y;  // throws ClassCastException
      return result;
   }
   // other methods omitted
}
   //Test: Truck suv = new Truck(4); Truck co = suv.clone()
   //      Expected: suv.x = co.x; suv.getClass() = co.getClass()

Note: Relevant to Bloch, Item 11 page 54.
Book website: Vehicle.java, Truck.java, CloneTest.java
```

(a) Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.

(b) If possible, give a test case that does not execute the fault. If not, briefly explain why not.

(c) If possible, give a test case that executes the fault, but does not result in an error state. If not, briefly explain why not.

(d) If possible give a test case that results in an error, but not a failure. If not, briefly explain why not. Hint: Don't forget about the program counter.

(e) In the given code, identify the first error state. Be sure to describe the complete state.

(f) Implement your repair in and verify using JUnit that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

3. Assume we have a class to compute N prime numbers. It takes one input, an integer N, and creates a list of N numbers where each number is prime (starting with 2).

   Assume the following methods:
   • computePrimes (int N)
   • int getFirstPrime()
   • int getNextPrime()
   • String toString() Returns the string in the form: "[2, 3, 6]"
   Assume the class has a fault that causes it not to include prime numbers whose last digit is 9 (for example, it omits 19, 29, 59, 79, 89, 109, ...).

   If possible, describe the following five tests in an abstract way. You can describe the tests as sequences of calls to the above methods, or briefly describe them in words.

   (a) A test that does not reach the fault

   (b) A test that reaches the fault, but does not infect

   (c) A test that infects the state, but does not propagate

   (d) A test that propagates, but does not reveal

   (e) A test that reveals the fault If a test cannot be created, explain why.