SFWRENG 3S03 Software Testing (2025 Winter)
Assignment 1

Administrative information

Weight of the assignment: this assignment is worth 10% of your final grade. It consists of three questions for a total of 70 marks, and a quality multiplier (1.0-0.5) that reflects the overall quality of your report.

How to complete the assignment?

- Answer all questions.
- Submit your solutions on Avenue.
- Only submit your report with your answers as a single PDF document.
- In your PDF document, report the key points and the key artifacts (JUnit tests, ChatGPT prompts, etc). Argue your points when asked to.

Groupwork

You can work in a group of up to three people, although you don't have to.
If you work in a group: everyone in the group needs to submit the very same PDF file (due to administrative reasons).

Due date: February 10, 2025, at 11:59 PM ET.

1. Testing with generative AI [10 marks]

Suppose you are developing a game. This action-adventure game involves exploration in a post-apocalyptic version of McMaster University, where up to eight players may be accosted by zombies, mutated professors, angry Deans, killer squirrels, and more. Gameplay can be both competitive and cooperative. The game is to be released on the PS5, Xbox Series X, Nintendo Switch and, for some reason known only to your designers, the Sega Genesis, which is 16-bit only and hasn't been manufactured since 1997.

a) [*3 marks*] Ask your favourite generative AI technology (e.g., ChatGPT, Bard) to specify three types of tests for such a game. Clearly state both the prompt(s) you used, and the response the generative AI provided.

b) [*2 marks*] State two things that you think are poor, incorrect, or unclear about the answers given by the generative AI technology in (a). For each, state why you think it's unhelpful.

c) [*2 marks*] State two things that you think are good, valid, or helpful about the answers given by the generative AI technology in (a). For each, state why you think it's helpful.

d) [*3 marks*] Based on what you have learned from this, *without using generative AI*, state three tests for the game briefly outlined above. In at most two sentences, explain why you prefer these tests to the ones presented in (a).

2. Testing with JUnit [40 marks]

Below you'll find four programs, each containing fault (or defects). Each also includes test inputs that will result in a failure.

Program 1:

```
/**
    * 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 static 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
```

Program 2:

```
   /**
    * Finds last index of zero
    *
    * @param x array to search
    *
    * @return last index of 0 in x; -1 if there is no zero
    * @throws NullPointerException if x is null
   */
```

```
   public static int lastZero (int[] x)
   {
      for (int i = 0; i < x.length; i++)
      {
         if (x[i] == 0)
         {
            return i;
         }
      }
      return -1;
   }
// test: x=[0,1,0]; Expected = 2
```

Program 3:

```
/**
   * Counts positive elements in array
   *
   * @param x array to search
   * @return number of positive elements in x
   * @throws NullPointerException if x is null
   */
   public static int countPositive (int[] x)
   {
      int count = 0;

      for (int i=0; i < x.length; i++)
      {
         if (x[i] >= 0)
         {
            count++;
         }
      }
      return count;
   }
      // test:  x=[-4, 2, 0, 2]
      // Expected = 2
```

Program 4:

```
/**
   * Count odd or positive elements in an array
   *
   * @param x array to search
   * @return count of odd or positive elements in x
   * @throws NullPointerException if x is null
   */
   public static int oddOrPos (int[] x)
   {  // Effects:  if x is null throw NullPointerException
      // else return the number of elements in x that
      //      are either odd or positive (or both)
      int count = 0;

      for (int i = 0; i < x.length; i++)
      {
         if (x[i]%2 == 1 || x[i] > 0)
         {
```

```
            count++;
         }
      }
      return count;
   }
      // test:  x=[-3, -2, 0, 1, 4]
      //        Expected = 3
```

Answer the following questions about each program. (The mark allocation below is specified *per program*; that is, your answers are worth 9 marks per program for a total of 36 marks for this question. Be sure to remember the definitions of failure, fault and error given in lecture.)

a) [*3 marks*] Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
b) [*2 marks*] If possible, give a test case that does not execute the fault. If this is not possible, briefly explain why.
c) [*1 mark*] If possible, give a test case that executes the fault but does not result in an error state. If this is not possible, briefly explain why.
d) [*1 mark*] If possible, give a test case that results in an error state, but not a failure. If this is not possible, briefly explain why. (Hint: don't forget about the program counter.)
e) [*3 marks*] For the test case given with the program, describe the first error state that's reached.

3. Test-driven development (TDD) [20 marks]

You are developing a simple calculator software, following test-driven principles. The next functionality you need to add is the `divide()` method with the usual semantics. Execute the main steps of TDD, in at least two iterations.

a) [*4 marks*] Develop a reasonable test suite to start the development of the functionality. Define at least two unit tests.
b) [*2 marks*] Develop the functionality until it passes the tests.
c) [*4 marks*] Is the functionality complete? Do you notice anything missing? Hint: compare the functionality to the code you would have written in a non-TDD way. Elaborate on this in detail.
d) [*4 marks*] Improve the test suite based on your previous answer.
e) [*2 marks*] Improve the functionality.

f) [*4 marks*] Would you use TDD? If yes: in what context? If no: why? What is the element you liked in TDD and what is the element you find the hardest to manage?

If you work in a team, try distributing these steps and observe how tests act as contracts. For example, let one person define the tests in (a), and someone else write the code in (b). Then a third person judge the code in (c) and the same person define the tests in (d). Finally, someone else improve the code in (e).

4. Overall quality of the report

Your report should be of reasonable quality. Your intent should be to report your experiences and argue your points, not to copy-paste template text.
To obtain your final mark, the sum mark of Tasks 1-2-3 will be scaled as follows.

Quality	Example characteristics	Multiplier
Good	Concisely and precisely reports artifacts, results, and if the apply, assumptions and limitations. Clearly argues points. Figures and images are of proper resolution. Uses proper English.	1.0
Medium	Reports artifacts and results at an acceptable level. Makes points without solid arguments. Awkward and hard-to-follow figures and images. Typos in text.	0.75
Poor	Clearly lacks effort at producing a quality report. Only reports data; interpretations and arguments are completely missing. Figures and images are of poor resolution. Poor English.	0.5