University of Sheffield

Data Provided: None

DEPARTMENT OF COMPUTER SCIENCE **Spring Semester 2022-2023**

SOFTWARE TESTING AND ANALYSIS **2 hours**

Answer THREE questions.

All questions carry equal weight. Figures in square brackets indicate the percentage of available marks allocated to each part of a question.

THIS PAGE IS BLANK.

1. This question involves a Java method called `crossover`. This method takes two strings as input parameters, `str1` and `str2`. It chooses a position at random (line 3) to swap the tails of the two strings (the part of each string after and including the randomly-chosen position) on lines 5 and 6, and returns the two newly created strings in an array (line 7).

```
1  public static String[] crossover(String str1, String str2, Random random) {
2      int maxPos = Math.min(str1.length(), str2.length());
3      int pos = random.nextInt(maxPos);
4      String[] result = new String[2];
5      result[0] = str1.substring(0, pos) + str2.substring(pos);
6      result[1] = str2.substring(0, pos) + str1.substring(pos);
7      return result;
8  }
```

For example, suppose `crossover` is called with the strings `"bold"` and `"test"`, and it randomly chooses position 1 to swap the tails of the two strings. In this example, `crossover` would return the array {`"best"`, `"told"`}.

The final parameter to the method is an object of class `Random`. This is a class that can return random integers via the `nextInt` method (called on line 3). The integer returned is greater than or equal to `0` and less than the supplied parameter. So `nextInt(5)` would return a random number in the range 0–4.

a) (i) Explain the similarities and differences between unit tests, integration tests and system tests.

(ii) Explain whether unit, integration, or system testing is the most appropriate for testing the `crossover` method, and why. Be sure to reference each type of testing in your answer, including your reasoning behind why it is/isn't appropriate.

[25%]

b) (i) Explain the similarities and differences between flaky and brittle tests.

(ii) Explain both *why* and *how* directly testing the `crossover` method — i.e., without doing anything to control the generation of random numbers — would likely result in a flaky test.

[25%]

c) Compare and contrast the different types of test double, explaining whether each different type of test double would be suitable for use in place of `Random` to avoid non-deterministic behaviour during testing of crossover.

[25%]

d) Write the code for a JUnit test to test the basic tail-swapping behaviour of `crossover`. The test should use a test double for `Random` to ensure deterministic behaviour. The test should supply `crossover` with the strings `"software testing"` and `"automate code"`. Together with the double, the test method should check that `crossover` returns the strings `"software code"` and `"automate testing"`. You can use calls to Mockito to simulate the test double, or alternatively, you can write your own test double class. You can omit `import` statements.

[25%]

2. The Java class `MyStack` implements a simplistic Stack data structure using an array, through the instance variable `elements`. `MyStack` has a further instance variable, `size`, that keeps track of the size of the stack:

```
 1  public class MyStack {
 2      Object[] elements;
 3      int size;
 4
 5      public MyStack(int maxLength) {
 6          size = 0;
 7          elements = new Object[maxLength];
 8      }
 9
10      public void push(Object obj) {
11          if (!full()) {
12              size ++;
13              elements[size] = obj;
14          }
15      }
16
17      public Object pop() {
18          if (empty()) {
19              return null;
20          }
21          size --;
22          return elements[size];
23      }
24
25      public boolean empty() {
26          return size == 0;
27      }
28
29      public boolean full() {
30          return size == elements.length;
31      }
32  }
```

The `push` method uses the `size` variable as the index to the `elements` array (line 13) to add an object to the stack. However, the `push` method contains a defect. It mistakenly increments `size` (line 12) *before* it puts the object into the array on line 13, instead of after. This means the "pushed" object is actually added to the array one index higher than it should be.

Because elements are placed into the `elements` array incorrectly, the `pop` method returns the wrong element. Instead of returning the element last pushed to the stack, it returns the one that was added before it.

If lines 12 and 13 were swapped, so that `size` was incremented after the statement "`elements[size] = object`", then the `push` method would work correctly, and the `pop` method would return the last pushed element, instead of the one that immediately preceded it.

MyStack has a JUnit test class MyStackTest, which has two tests:

```
public class MyStackTest {
    @Test
    public void testPush() {
        MyStack s = makeStack();
        assertThat(s.size, equalTo(2));
    }

    @Test
    public void testPop() {
        MyStack s = makeStack();
        assertThat(s.pop(), equalTo(1));
    }

    MyStack makeStack() {
        MyStack s = new MyStack(10);
        s.push(1);
        s.push(1);
        return s;
    }
}
```

a) (i) Defects, infections, and failures are different ways to characterise bugs. Explain the difference between the three terms.

(ii) In terms of the RIPR model explain why the testPop JUnit test does not fail, even though it exercises the defective push method, followed by the pop method, and involves an assertion on the object that pop returns.

[20%]

b) Ignoring the fact that MyStack's test suite does not reveal the defect in the push method, give up to FOUR different ways in which the *existing* tests in MyStackTest could, in general, be made clearer and more concise to other developers, or be made less prone to brittleness. Fully explain your reasoning in each case.

[20%]

c) Draw the Class Control Flow Graph (CCFG) for MyStack.

[20%]

d) i) What is a branch?

ii) Identify a branch in your drawn CCFG.

iii) What is the branch coverage of MyStack by the two tests in MyStackTest? Explain how you worked out your answer.

[20%]

e) Discuss why obtaining 100% coverage of some code by a test suite is not a guarantee that the code covered will be bug-free.

[20%]

3. This question concerns program slicing.

a) Define what it means for a program s to be a static backward end slice of a program p with respect to a variable y. [10%]

b) Consider the following fragment of code in which input(x) and input(y) input values from an interface or file and assign them to x and y respectively. Line numbers are included to help you explain your answers.

```
1.  input(x);
2.  input(y);
3.  z:=y+10;
4.  if (z<0){
5.      z:=-z;}
6.  while (z < 70) {
7.      if (z>30)
8.          x:=x+z;
       else
9.          x:=x-z;
10.     y:=y+z;
11.     if (z>20)
12.         z:=z+5;
       else
13.         z:=z+10;}
```

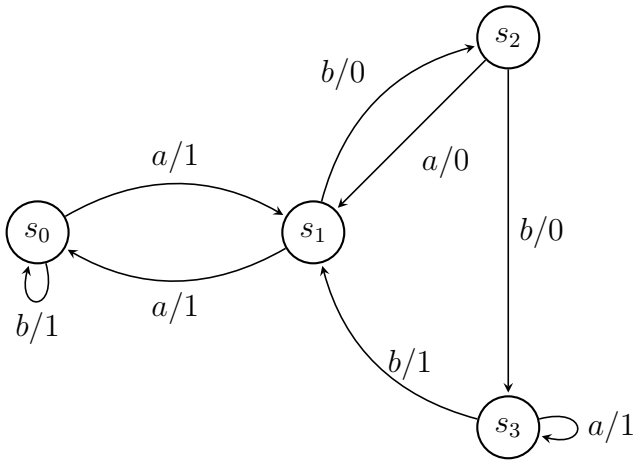
Produce the Program Dependence Graph for this fragment of code, *explaining* your answer. Note, there is no need to include edges that are self-loops.

[30%]

c) Consider again the fragment of code above. Produce the *minimal* static backward end slice of this fragment of code with respect to variable z. Make sure that you *justify* your answer: a) if you remove a statement then explain why you can remove it; b) if you keep a statement then explain why. Marks will be reduced if you do not justify your answer in terms of the PDG. [30%]

d) Explain what it means for a program s to be a static forward slice of a program p with respect to a variable x and the start of the program. As part of your answer you should give the minimal static forward slice the fragment of code above with respect to x. [30%]

4. This question will use the Finite State Machine (FSM) below, which will be referred to as M_0. Note that s_0 is the initial state of M_0.



a) Determine whether:

1. M_0 is completely-specified;
2. M_0 is deterministic;
3. M_0 is strongly-connected.

In each case explain what the property considered means (define the property) and justify your answer. [30%]

b) For each state s_i of M_0, determine whether s_i has a unique input output sequence (UIO), justifying your answer by either giving such a sequence or explaining why there is no such sequence.

[40%]

c) Explain how a UIO can be used to test a transition of an FSM. You should illustrate your answer by showing how a UIO can be used to test the transition in M_0 that has starting state s_3 and input a. [30%]

END OF QUESTION PAPER