Data Provided: None

DEPARTMENT OF COMPUTER SCIENCE SPRING SEMESTER 2023–2024

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

The first two questions of this exam refer to the Java method `sumDigits`, which appears below. The method takes a string of digits, and sums each digit to produce a total. Any non-digit character in the string is ignored:

```java
public static int sumDigits(String str) {
    int sum = 0;
    for (int i = 0; i < str.length(); i++) {
        char currentChar = str.charAt(i);
        if (Character.isDigit(currentChar)) {
            sum += Character.getNumericValue(currentChar);
        }
    }
    return sum;
}
```

1. a) Give THREE reasons, discussed in lectures, why finding all bugs for non-trivial computer programs is very difficult, and in general, impossible. Pick ONE reason and illustrate it with an example. [40%]

   b) This question requires you to insert ONE defect into the `sumDigits` method. Before doing so, ensure you have read the remaining parts of this question, so that you can be certain you have understood what kind of defect you need to introduce to be able to give an answer to each part.

      (i) Write out the `sumDigits` method with the defect you have introduced, clearly indicating the change you have made.

      (ii) Explain how the defect could result in a failure.

      (iii) Write a JUnit test case that would be capable of revealing the failure in the general scenario you described in part (ii).

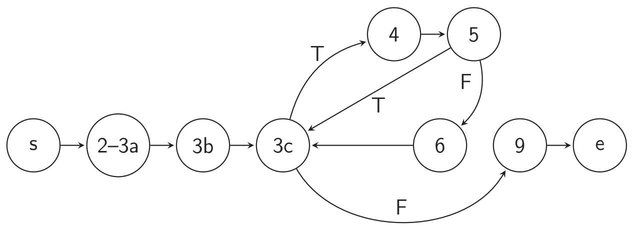
      (iv) Explain how the defect could result in an infection that *does not* propagate to the output, thereby not causing a failure.

      (v) Write a JUnit test case that would be capable of causing an infection but does not cause a failure according to the general scenario you described in part (iv).

      [60%]

2. The `sumDigits` method is a small but critical part of a larger payments system that José is writing.

   a) To help José test his application, Phil writes a utility called `CFGVisualiser` to visualise the control flow graph (CFG) of a given Java method. Unfortunately, Phil's program is buggy and it produced the following output for `sumDigits`:



   Describe TWO distinct aspects of the CFG that are incorrect.

   As part of your answer, list the individual edits you would need to make to change this incorrect graph into the correct one.

   [30%]

   b) One of Phil's JUnit tests for `CFGVisualiser`, `testHasEdge`, is shown below. It is part of a test suite that tests a class called `CFG`, which allows nodes to be added with a string ID (via the public method `void addNode(String nodeID)`) and edges to be assigned between nodes (via the public method `void addEdge(String sourceNodeID, String targetNodeID)`). The public method `boolean hasEdge(String sourceNodeID, String targetNodeID)` returns `true` if an edge exists between two nodes, and false otherwise.

```
@Test
public void testHasEdge() {
    CFG cfg = makeCFG();
    assertTrue(cfg.hasEdge("s", "1"));
    assertFalse(cfg.hasEdge("2", "3"));
}
```

   Describe THREE things Phil could do to improve his test so that its intent and meaning are clearer to other developers.

   [30%]

   c) Meanwhile, José is testing his payments system and the `sumDigits` method. One of his tests includes the line `sumDigits("xy")`.

      (i) What is the path taken through the CFG of `sumDigits` with this particular string input? Write out the sequence of nodes executed.

      (ii) Assuming this is the only call to `sumDigits` in the test, what is its statement coverage? Make sure you show all of your working.

   [20%]

   d) José has another test for `sumDigits` that covers 50% of its statements. Which is the better test, this one or the one from part (c)? Explain your reasoning.

   [20%]

3. a) Briefly explain the following concepts:

   (i) *Equivalent mutants* and their impact on mutation testing.

   (ii) *Regression testing* and the context in which it can be a suitable technique.

   (iii) *Fuzzing*, including its advantages and disadvantages.

   (iv) *Search-based test generation*, in contrast with random testing.

[40%]

b) Consider the following Java program and corresponding test coverage matrix:

```
 1  int power(int base, int exponent){
 2      if (exponent < 0)
 3          throw new Exception("Negative exponent");
 4      if ((base == 0) && (exponent == 0))
 5          throw new Exception("Undefined");
 6      int result = 1;
 7      while (exponent > 0){
 8          result *= base;
 9          exponent--;
10      }
11      return result;
12  }
```

Test	Inputs	Testing requirement (lines of code)								
		2	3	4	5	6	7	8	9	11
t_1	(0, -10)	x	x							
t_2	(0, 0)	x		x	x					
t_3	(10, 0)	x		x		x	x			x
t_4	(1, 1)	x		x		x	x	x	x	x
t_5	(2, 2)	x		x		x	x	x	x	x
t_6	(-10, 0)	x		x		x	x			x
t_7	(-2, 2)	x		x		x	x	x	x	x

(i) Apply a test minimisation algorithm to find a minimal equivalent test suite, explaining your workings.

(ii) Assume that a change has been made in the program whereby the statement in line 9 has been mistakenly removed, and the statement in line 5 has been changed to throw a different exception. In this scenario, apply an intraprocedural test selection algorithm, explaining your workings.

(iii) Using your own fictitious fault detection data for each test (e.g., from a total of 31 mutants, t_1 kills 4, t_2 kills 1, etc, with at most two tests killing the same number of mutants), apply a test prioritisation algorithm to identify the best test execution order, explaining your workings.

[60%]

4. a) Briefly describe Model-based Testing and discuss its benefits and challenges as a testing technique. [20%]

b) Briefly explain what it means for a Finite State Machine (FSM) to be:

(i) *deterministic*

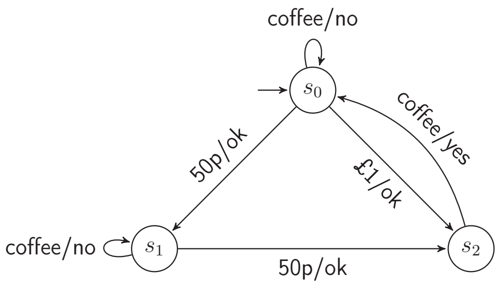
(ii) *minimal*, and

(iii) *completely specified*

Use your own illustrative example to support your explanations.

[20%]

For the next two question parts, consider the following state machine model of a simple coffee machine system:



c) You decide to apply the W Method.

(i) Give a state cover V for the FSM, justifying your answer.

(ii) Give a characterising set for the FSM, justifying your answer.

[30%]

d) You are asked to evolve the system in the following ways:

- The cost of a coffee increases to £2.
- *Only* £1 and £2 coins are accepted.
- If the user inserts more money than the cost of the coffee, the machine should output the change immediately, e.g., "£X/£Y" indicates that the user inserted £X and the machine output £Y.
- Whenever the machine is holding any amount of money, it should be possible to input *refund*. The machine should then allow the user to input either *confirm*, in which case the machine should output the full amount of money held and transition to its initial state, or *cancel*, in which case the machine should simply transition back to its previous state (i.e., before the *refund* input).

Draw the resulting FSM, explaining how your changes fulfil the new specification.

[30%]

END OF QUESTION PAPER