MTAT.03.159 – Software Testing

Written Exam – 29 May 2014

Important Notes:

- The exam is open-book and open-laptop. Web browsing is allowed, but you are not allowed to use e-mail clients or Instant Messaging clients, or to share any information "live" with anybody inside or outside the exam room during the time of the exam.
- At the end of the exam you must submit both the question sheets and your answer sheets. To avoid that any of your solutions get lost, make sure to write your name and student ID on each sheet of paper that you submit. Also, please number the pages on your answer sheets.
- Write clearly. Answers that are illegible cannot be counted as correct answers. Only answers written in English will be marked.
- Total marks: 40 (equivalent to 40% of total grade).

==

Task 1 (7 marks):

According to international standards, an 'error' is a mistake (a misunderstanding, a misinterpretation, a mix-up, etc.) occurring in the mind of a person, a 'fault' (or 'defect' or 'bug') is an incorrect statement in a document (e.g., requirements specification or code), and a 'failure' is incorrect/unintended behaviour of software when executed.

The relations between 'error', 'fault' and 'failure' can be expressed as follows:

> *A programmer made an error which resulted in a fault in the program code, which triggered a failure when executed.*

a) For each of the following statements, decide whether it describes an 'error' (1), a 'fault' (2), a 'failure' (3), or 'something else' (4).

- Statement A: A programmer interpreted an interface specification incorrectly.
- Statement B: A program statement uses the operator '+', while using '-' would have been correct.
- Statement C: A program crashes when a tester enters an invalid input.
- Statement D: A program crashes when a tester enters a valid input.
- Statement E: A reviewer reports that he/she found a defect in a requirements spec. Later it turns out to be a 'false positive'.

For each statement, justify your answer choice with an argument. Try to be precise and concise.

(5 marks)

b) Can *errors* be detected by a C++ compiler? Justify/Motivate your answer. You don't get marks for a simple 'yes' or 'no'.

Hint: Use the definition of 'error' as stated above in the task description!

(1 mark)

c) Can *failures* be detected by a C++ compiler? Justify/Motivate your answer. You don't get marks for a simple 'yes' or 'no'.

Hint: Use the definition of 'Failure' as stated above in the task description!

(1 mark)

Task 2 (3 marks):

The table below characterises the terms 'verification and 'validation'.

Term	Characterisation
Verification	To check whether the software has been done right (i.e., software is working as specified)
Validation	To check whether the right software has been developed (i.e., software is working as needed/expected by customer)

Decide, whether the following statements are true or false. Explain/Justify your answers using the characterisations provided in the table above.

- Statement A: *Requirements inspection* is a validation activity. → true or false?
- Statement B: *Unit testing* is a verification activity. → true or false?
- Statement C: *Code inspection* is a validation activity. → true or false?

(3 marks)

Task 3 (4 marks):

Briefly describe the essentials of *white-box* and *black-box testing*. In addition, give two examples for *white-box* and two examples of *black-box testing strategies*.

Hint: Test levels are not examples of testing strategies.

(4 marks)

Task 4 (6 marks):

Consider a software module that is intended to accept the name of a grocery item, *gname*, and a list of the different sizes the item comes in, *gsize1* to *gsize5*, specified in litres. A maximum of five sizes may be entered for each item.
The specification states that the item name is to be entered first, followed by a comma, then followed by a list of sizes. A comma must be used to separate each size. Spaces (blanks) are to be ignored by the software anywhere in the input. The input must be entered in one line. The item name is to be alphabetic characters with a length of 2 to 15 characters. Each size may take a value in the range of 1 to 10, whole numbers (integers) only. The sizes are to be entered in ascending order (smaller sizes first).

Based on this specification, many Equivalence Classes for testing the software module can be derived. The following list shows some examples:

1. Item name is alphabetic (valid)
2. Item name is not alphabetic (invalid)
3. Size value is less than 1 (invalid)
4. Size value is in the range 1 to 10 (valid)
5. …

Add 6 additional equivalence classes (comprising at least 2 invalid input classes) and provide a set of test cases that cover the 10 classes listed (i.e., the 4 classes above and your 6 classes). Try to make the set of test cases as small as possible. Remember that a complete test case also contains an output value. To make things easy, assume that there are only two output values, i.e., 'input accepted' and 'invalid input', for valid and invalid inputs, respectively.

Hint: First list the equivalence classes then list the test cases. Say for each test case which equivalent class(es) have been covered. (6 marks)

Task 5 (12 marks):

For the method `speedingfine` below, perform tasks a), b) and c):

```
1  public static int speedingfine (int age, int overspeed; int
licencemark) {
2    int fine = 0;
3    if ((age >= 25) && (overspeed < 30) && (licencemark < 3))
4       fine = fine + 100 * overspeed;
5    else {
6      if ((age < 25) || (licencemark >= 3))
7        fine = fine + (200 * overspeed);
8      if (overspeed >= 30)
9        fine = fine + 5000;
10   }
11   return fine;
12 }
```

a) Draw the control flow graph and calculate the McCabe Cyclomatic number (i.e., the number of linearly independent paths)

(2 marks)

b) Write down a minimal set of test cases needed to achieve 100% statement coverage.

(4 marks)

c) 100% Decision/Condition (D/C) coverage requires that all decisions are evaluated at least once to ‘true’ and once to ‘false’, and that all atomic conditions (within a decision) are evaluated at least once to ‘true’ and once to ‘false’. Write down the test cases needed in addition to those listed under b) to achieve 100% D/C coverage. Show how each test case in b) and c) satisfies the D/C criterion, i.e., say for each test case how each decision and each atomic condition is evaluated.

(6 marks)

Hint 1: Remember that complete test cases include both input values and expected output values.
Hint 2: An example of an atomic condition is ‘overspeed < 30’, contained in the first decision (line 3).

<u>Task 6 (5 marks):</u>

An simple editor may have four input parameters with values as shown below. Note that one parameter has 3 values while the other three parameters have only 2 values.

Parameter	Value 1	Value 2	Value 3
Font type	Times	Arial	Cambria
Font size	large	small	
Font style	normal	bold	
Font colour	black	red	

a) If you wanted to test all possible combinations of parameter values (i.e., the number of all 4-way interactions), how many test cases would you need? Show your calculation.

(1 mark)

b) How many pairwise interactions (i.e., 2-way interactions) between parameter values exist? Show your calculation.

(1 mark)

c) Show that you can cover all 2-way interactions between parameter values with only 6 test cases. Present the set of test cases (per test case you need to specify only the values of the input parameters)

(3 marks)

<u>Task 7 (3 marks):</u>

Three reviewers have inspected a document and found the defects shown in Table 1 below.
Use a capture-recapture model to estimate the number of <u>remaining</u> defects in the document. You must show the details of your calculation to get marks.

Defect	Reviewer 1	Reviewer 2	Reviewer 3
D1	1	1	0
D2	1	0	0
D3	0	1	1
D4	1	0	1
D5	1	1	0
D6	1	0	0
D7	0	1	1
D8	1	0	1
D9	1	1	0
D10	1	0	0
D11	0	1	1
D12	0	0	1
D13	0	0	1

(3 marks)