• The exam is open-book and open-laptop. Web browsing is allowed, but you are neither allowed to use e-mail clients or Instant Messaging clients nor to share any information “live” with anybody inside or outside the exam room.

• This document (question sheet) contains 6 pages (including the cover page). Please check that you have received 6 pages.

• 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.

• Write clearly. Answers that are illegible cannot be counted as correct answers. Only answers written in English will be marked.

• To answer Part 1, use the separately distributed answer sheet. Answers given on the question sheets will not be marked!

• To answer Part 2, use the separately distributed blank paper. Answers given on the question sheets will not be marked! Also, please number the pages on your answer sheets.

• At the end of the exam you must return the problem sheet. If you take the question sheets with you (out of the exam room), this will be considered academic fraud (cheating) and treated accordingly.

• Total marks: 30 (equivalent to 30% of the total grade). You must get at least 8 marks in this exam – otherwise you fail the course no matter how many marks you received in the lab assignments.

===========================================================================

PART 1: Multiple-Choice Questionnaire (8 marks)

PART 2: Open Questions & Constructive Tasks (22 marks)

---

Total: 30 marks
PART 1: Multiple-Choice Questionnaire (8 marks)

Important: For Part 1, please check boxes on the separate questionnaire answer sheet. Read carefully before you answer and observe instructions carefully!

The following questions (up to question Q-08) have exactly one correct answer, thus, you must check exactly one answer box on the separate answer sheet. If you think that more than one answer is correct, choose the one answer that seems to be most correct/suitable/relevant.

Have a look at the following code example and then answer question Q-01.

```java
public static int numZero (int[] x) {
    // Effects: if x==null throw NullPointerException
    // else return the number of occurrences of 0 in x
    int count = 0;
    for (int i = 0; i < x.length; i++) {
        if (x[i] == 0) {
            count++;
        }
    }
    return count;
}
```

Q-01 (1 mark): Assume the input to numZero() is x=[3, 8, 1]. Which of the following statements is correct?

Answer choice:
A: The program is faulty and will trigger a failure.
B: The program will not run into an error state.
C: The program does not contain a fault and count equals 0 when the program stops.
D: The program runs into an error state but will not trigger a failure.

Q-02 (1 mark): Assume the following if-statement: if (x < 6 & y >4) then ... //do something
Which of the following test data assignments for x and y before running into the if-statement would be a counter-example to the (wrong) statement ‘(Simple) Condition Coverage’ subsumes ‘Decision Coverage’?

Answer choice:
A: Test data 1: x = 0, y = 10 / Test data 2: x = 0, y = 0
B: Test data 1: x = 0, y = 0 / Test data 2: x = 10, y = 10
C: Test data 1: x = 0, y = 0 / Test data 2: x = 0, y = 10
D: Test data 1: x = 0, y = 10 / Test data 2: x = 10, y = 10

Q-03 (1 mark): Which of the following activities is a validation activity?

Answer choice:
A: Design inspection (done by architect and developers)
B: Acceptance testing (done by end users)
C: Code inspection (done by testers and developers)
D: Unit testing (done by developers)
Q-04 (1 mark): A state-transition can be expressed as follows: \( S_i \rightarrow E_m/A_n \rightarrow S_j \), where \( S_i \) is the i-th state, \( S_j \) is the j-th state (with j not necessarily different from i), \( E_m \) is the m-th event (input) and \( A_n \) is the n-th action (output). Which of the following statements regarding the strength of State-Transition testing coverage criteria is correct?

Answer choice:
A: Covering all events is at least as strong a criterion than covering all state-transitions.
B: Covering all states is at least as strong a criterion than covering all state-transitions.
C: Covering all state-transitions is at least as strong a criterion than covering all events.
D: Covering all states is at least as strong a criterion than covering all events.

Q-05 (1 mark): In the context of mutation testing, assume that in your Java program you replace the calculation \( y=2\times x \) by \( y=2; \ x \), to create a mutant. What kind of mutant would you create?

Answer choice:
A: Trivial mutant
B: Equivalent mutant
C: Stillborn mutant
D: Erroneous mutant

Q-06 (1 mark): You plan to test the correct functioning of a program with four simultaneous inputs. Each input can be either ‘0’ or ‘1’ (i.e., it’s a Boolean). Which statement is correct?

Answer choice:
A: Complete testing of all possible combinations of inputs requires 24 test cases
B: Complete pairwise testing cannot be done with less than 9 test cases
C: The number of pairwise interactions between input values is 24
D: Complete testing of all 4-way interactions requires more than 16 test cases

Q-07 (1 mark): Which statement about Data-Driven Testing (DDT) is not correct?

Answer choice:
A: DDT is the execution of the same set of test steps with multiple (different) data
B: In DDT, test data (i.e., inputs and expected outcomes) are integrated with the test scripts
C: In DDT, test data (i.e., inputs and expected outcomes) for test cases are read from external data sources
D: In DDT it is easy to maintain test data and test scripts by different roles (e.g., customer representatives and professional testers, respectively)

Q-08 (1 mark): Three reviewers (Rev1, ..., Rev3) inspected a document and found the defects {D1, ..., D10} shown in the table below. A defect found by a reviewer is marked with ‘x’.

<table>
<thead>
<tr>
<th></th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
<th>D4</th>
<th>D5</th>
<th>D6</th>
<th>D7</th>
<th>D8</th>
<th>D9</th>
<th>D10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev 1</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Rev 2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Rev 3</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the number of remaining faults if you use the capture-recapture model \( M_t \) for estimation? Note: apply standard rounding (i.e., don’t truncate).

Answer choice:
A: 11
B: 2
C: 1
D: 0
PART 2: Open Questions & Constructive Tasks (22 marks)

**Task 1 – Functional Testing (10 marks):**

Figure 1 shows the GUI of an application called ‘Stack’.

![Stack App GUI](image)

The requirements list for ‘Stack’ states the following:

**R1:** When the ‘Stack’ App is switched on, the input field ‘Value’ will show ‘0’, the output fields ‘Top-Value’ and ‘Message’ will be empty, and the Stack Pointer (SP) will be set to -1. Note: The internal variable SP is not visible to the user and cannot be directly accessed by the user.

**R2:** The Stack array has 2 elements and is neither visible to nor directly accessible by the user.

**R3:** Input field ‘Value’: valid inputs are natural numbers greater than 0; all other inputs are invalid.

**R4:** The user can either press ‘Pop’ or ‘Push’ (radio button). As soon as either ‘Pop’ or ‘Push’ is pressed, Output is generated. Note: When pressing ‘Pop’ whatever has (or has not) been entered in field ‘Value’ will be ignored.

**R5:** If SP > -1 and ‘Pop’ is pressed, then the output field ‘Message’ will show ‘POP’ and the output field ‘Top-Value’ will show the value in the stack element to which SP is pointing; then SP will be reduced by 1.

**R6:** If SP = -1 and ‘Pop’ is pressed, then the output field ‘Message’ will show ‘STACK EMPTY’ and the output field ‘Top-Value’ will be empty; The value of SP will not change.

**R7:** If ‘Push’ is pressed and the field ‘Value’ has received invalid input, then the output field ‘Message’ will show ‘INVALID INPUT’ and the output field ‘Top-Value’ will be empty. The value of SP will not change.

**R8:** If SP < 1 and ‘Push’ is pressed (and the field ‘Value’ has received valid input), then the output field ‘Message’ will show ‘PUSH’ and the output field ‘Top-Value’ will show the input received in field ‘Value’. The value of SP will be increased by 1.

**R9:** If SP = 1 and ‘Push’ is pressed and the field ‘Value’ has received valid input, then the output field ‘Message’ will show ‘STACK FULL’ and the output field ‘Top-Value’ will be empty. The value of SP will not change.

**To Do (continues on next page):**

a) [2 marks] Based on R1-R9, define suitable equivalence classes for each input & output variable.

b) [4 marks] Create a state-transition diagram that covers all requirements R1 to R9. The state-transition diagram should show all states and all state transitions, and a state transition shall be represented by an arrow annotated with information about the inputs that trigger the transition and the expected outputs that are triggered by the transition.

*Hint: Use the possible values of SP to define the states of the stack; this will give you 3 states.*

c) [1 mark] Assume that a test report contains the following data (test result):

Value=1, RadioButton=Push, Top-value=1, Message=PUSH.

Does the test report show a failure? Justify/explain your answer. Assume that the test code was correctly implemented and executed and the test result correctly reported.
d) [3 marks] Assume that the following test scenarios have been applied (after switching on the app):

\[ s_1 \rightarrow \text{Value='0' AND RadioButton=PUSH} \rightarrow s_1 \]
\[ s_1 \rightarrow \text{Value='1' AND RadioButton=PUSH} \rightarrow s_2 \rightarrow \text{RadioButton=POP} \rightarrow s_1 \]

Answer the following questions (with brief justifications/explanations):

d.1) What is the state coverage (in percent)?

d.2) Write down an additional test scenario that would cover the state transition from \( s_2 \) to \( s_2 \).

**Task 2 – Structural Testing (8 marks):**

For the method \( \text{evenSum} \) shown below, perform tasks a), b), c), d) and e). You can assume that \( \text{nums} \) has been well-defined (i.e., is not null and has integers) when the method \( \text{evenSum} \) is called.

```java
public static int evenSum(int[] nums) {
    int esum = 0;
    int i = 0;
    while (i < nums.length) {
        if(nums[i] % 2 == 0) {
            esum += nums[i];
        }
        i++;
    }
    return esum;
}
```

**To Do:**

a) [2 marks] Draw the control flow graph (CFG) and calculate the McCabe Cyclomatic number, i.e., the number of linearly independent paths. Show how you calculate the McCabe complexity. *Hint: Use the line numbers of the code snippet shown in the problem statement above to label the nodes in your control-flow graph (CFG).*

b) [1 mark] Write down a set of linearly independent paths. Use the code line numbers to express the paths.

c) [1 mark] Write down a minimal set of test cases needed to achieve 100% statement coverage. For each test case, state the path in the CFG that it covers. Remember that a complete test case consists of input and (expected) output values.

d) [2 marks] Write down a minimal set of test cases needed to achieve 100% coverage of the set of linearly independent paths shown in part b). For each test case, state the path that it covers.

e) [2 marks] Write down all DU pairs and all def-use paths of variable \( i \). Then define a minimal set of test cases needed to cover all def-use paths of variable \( i \). Say for each test case, which def-use path(s) it covers.

*Hints: There are in total eight DU pairs; the statement \( i++ \) contains a use (U) and a def (D) of variable \( i \); line 8 itself is not a def-use path (because it’s a node) but it is the start and end-point of several def-use paths.*
Task 3 – Fault, Error, Failure (4 marks):

Have a careful look at the following (faulty) program:

00 public int findLast (int[] x, int y)
01 {
02    // Effects:
03    // If x==null throw NullPointerException.
04    // Else return the index of the last element in x that equals y.
05    // If no such element exists, return -1.
06    int i = x.length - 1;
07    while (i > 0)
08        { //
09            if (x[i] == y)
10                {
11                    return i;
12                }
13            i--;
14        }
15    return -1;
16 }

To Do:

a) [1 mark] Where is the fault and how should it be corrected?

b) [1.5 marks] Assume, you apply the following test cases TC1 and TC2 to the original program shown above:

(TC1) input: x = [1, 4, 5]; y = 1 – expected output: 0

(TC2) input: x = [1, 4, 5]; y = 5 – expected output: 2

Say for each test case whether it triggers a failure. Justify your answers.

c) Using TC1 from part b), the program will execute the states shown in the table below.

Which state is the first error state? [0.5 marks]

Define a test case that runs into an error state but does not trigger a failure? [1 mark]

<table>
<thead>
<tr>
<th>State 0: x -&gt; [1, 4, 5] y -&gt; 1 i: undefined PC: line 00</th>
<th>State 1: x = [1, 4, 5] y = 1 i -&gt; 2 PC: line 07</th>
<th>State 2: x = [1, 4, 5] y = 1 i = 2 PC: line 08</th>
<th>State 3: x = [1, 4, 5] y = 1 i = 1 PC: line 10</th>
<th>State 4: x = [1, 4, 5] y = 1 i -&gt; 1 PC: line 14</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 5: x = [1, 4, 5] y = 1 i = 1 PC: line 08</td>
<td>State 6: x = [1, 4, 5] y = 1 i = 1 PC: line 10</td>
<td>State 7: x = [1, 4, 5] y = 1 i -&gt; 0 PC: line 14</td>
<td>State 8: x = [1, 4, 5] y = 1 i = 0 PC: line 08</td>
<td>State 9: x = [1, 4, 5] y = 1 i = 0 PC: line 16</td>
</tr>
</tbody>
</table>

Note: ‘PC’ stands for ‘Program Counter’; the PC points to the program statement that – after execution – will bring the program into the next state; the PC is not a program variable!