The exam is open-book and open-laptop. Web browsing is allowed but you are not allowed to do any of the following during the exam:

- Use e-mail clients
- Use Instant Messaging clients
- Use any other method of communication to exchange information with anybody inside or outside the exam room.

Any violation of the rules is considered cheating and automatically results in 0 marks for the exam. In addition, cheaters will be reported to the Head of Institute and the Dean's office.

At the end of the exam you must submit both the question sheets and your answer sheets. Note that answers given on the question sheet (i.e., this document) will not be considered for marking. To avoid that any of your solutions get lost, make sure to write your full 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 the total grade). You must get at least 10 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 (10 marks)
PART 2: Open Questions & Constructive Tasks (30 marks)

---

Total: 40 marks (≈100%)
PART 1: Multiple-Choice Questionnaire (10 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.

Q-01 (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-02 (1 mark): Which of the following statements describes a failure?
Answer choice:
A: A programmer misunderstood an interface specification
B: An expression in Java code uses the operator ‘-’, while using ‘+’ would have been correct
C: A program crashes when a tester enters a valid input
D: A reviewer reports that he/she found a defect in a requirements spec. Later it turns out to be a ‘false positive’

Q-03 (1 mark): Which of the following statements about the relationships between control-flow coverage criteria is correct?
Answer choice:
A: Condition Coverage subsumes Statement Coverage
B: Decision Coverage subsumes Statement Coverage
C: Decision Coverage subsumes Condition Coverage
D: Condition Coverage subsumes Condition/Decision Coverage

Q-04 (1 mark): Which of the following statements describes ‘Selective Regression Testing’ best?
Answer choice:
A: After any kind of change, all existing code will be re-tested
B: Only after a bug fix, all code will be re-tested
C: Only code affected by a code change will be re-tested
D: Only after a functional enhancement, all code will be re-tested

Q-05 (1 mark): Which statement about Data-Driven Testing (DDT) is not correct?
Answer choice:
A: In DDT, test data (i.e., inputs and expected outcomes) are integrated with the test scripts
B: DDT is the execution of the same set of test steps with multiple (different) data
C: In DDT, test data (i.e., inputs and expected outcomes) for all test cases are read from an external data source
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-06 (1 mark): Which of the following statements about test techniques is correct?
Answer choice:
A: Black-box testing techniques exploit knowledge about the code that is tested
B: Equivalence class partitioning is a white-box testing technique
C: Mutation testing does require a test oracle derived from the program specification
D: White-box testing techniques exploit knowledge about the code that is tested

Q-07 (1 mark): Assume you have a simple web-based calculator. The UI has 3 input fields: Operand1, Operator, and Operand2. Operand1 and Operand2 have each 9 valid inputs: {1, 2, ..., 9} and the Operator has 4 valid inputs: {+, -, *, /}. Assuming that you can only enter valid inputs in all three input fields: How many test cases does exhaustive functional testing require?
Answer choice:
A: 3
B: 22
C: 256
D: 324

Q-08 (1 mark): In the context of mutation testing, assume that in your program you replace the calculation \( y=2x \) by \( y=x+x \), to create a mutant. What kind of mutant would you create?
Answer choice:
A: Trivial mutant
B: Stillborn mutant
C: Equivalent mutant
D: Erroneous mutant

The following question (question Q-09) can have more than one correct answer. You must check all correct answer choices to get full marks. You get partial marks, if you check some of the correct answer choices. You will get a penalty, if you check an incorrect answer choice. You don’t get a penalty, if you miss a correct answer choice. Overall, the lowest possible mark you can get is 0 (i.e., even if everything you check is wrong, you won’t get a negative mark).

Q-09 (2 marks): 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(s) is (are) correct?
Answer choice:
A: Complete testing of all possible combinations of inputs requires 16 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
PART 2: Open Questions & Constructive Tasks (30 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: As soon as either button ‘Pop’ or button ‘Push’ is pressed, Output is generated. Note: When pressing the ‘Pop’ button whatever has (or has not) been entered in field ‘Value’ will be ignored.

R5: If SP > -1 and the ‘Pop’ button 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 the ‘Pop’ button 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 the ‘Push’ button is pressed and the field ‘Value’ has received invalid input, then the output field ‘Message’ will show ‘INVALID DATA’ and the output field ‘Top-Value’ will be empty. The value of SP will not change.

R8: If SP < 1 and the ‘Push’ button 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 the ‘Push’ button 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:

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

b) Create a statechart that covers all requirements R1 to R9. The statechart 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. [4 marks]

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

c) Create a table with test cases such that all state transitions in the statechart created in b) are covered at least once. The table shall contain the following information: current state, input1 value, input2 value, output1 value, output2 value, next state. [2 marks]

d) Assume that a test report contains the following data (test result):

   Value=1, Button=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 result correctly reported. [2 marks]
Task 2 – Structural Testing (10 marks):

For the method \( \text{xyzr} \) shown below, perform tasks a), b), c) and d). You can assume that \( x, y, \) and \( z \) are non-negative integers with \( x \) and \( y \) in the range \([1, 100]\) and \( z \) in the range \([0, 10]\).

```java
public static int xyzr (int x, int y; int z) {
    int r = 0;
    if ((x >= 25) && (y < 30) && (z < 3))
        r = r + 10 * y * (z + 1);
    else {
        if ((x < 25) || (z >= 3))
            r = r + 20 * y * (z + 1);
        if (y >= 30)
            r = r + 5000;
    }
    return r;
}
```

To Do:

a) 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. [3 marks]

Hint: Use the line numbers of the code example above to label the nodes in your control-flow graph (CFG). Also, the nodes in a CFG correspond to decisions in the code.

b) 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. [3 marks]

Hint: Remember that a test case consists of input and (expected) output values.

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

d) Write down a table that shows all def-use paths of variable \( x \). then define a minimal set of test cases needed to cover all def-use paths of variable \( x \). Say for each test case, which def-use path(s) it covers. [2 marks]

Task 3 – Review & Defect Estimation (4 marks):

Three reviewers have inspected a document and found the defects shown in the table below. A found defect is marked with ‘x’.

<table>
<thead>
<tr>
<th>Reviewer</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>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Rev 2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Rev 3</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First use a capture-recapture model to estimate the numbers of remaining defects in the document when using the data of reviewers Rev 1&2 only, Rev 1&3 only, and Rev 2&3 only. [1.5 marks]

Then use a suitable capture-recapture model and estimate the remaining defects for all three reviewers Rev 1&2&3. You must show the details of all your calculations to get marks. [2.5 marks]
Task 4 – Fault, Error, Failure (6 marks):

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

```java
public int findLast (int[] x, int y)
{
    //Effects: If x==null throw NullPointerException
    // else return the index of the last element
    // in x that equals y.
    // If no such element exists, return -1
    for (int i=x.length-1; i > 0; i--)
    {
        if (x[i] == y)
        {
            return i;
        }
    }
    return -1;
}
```

**To Do:**

a) Where is the fault and how should it be corrected? [2 marks]

b) Assume, you apply the following test cases TC1 and TC2:

   (TC1) input: x = [2, 3, 5]; y = 2 – expected output: 0
   (TC2) input: x = [2, 3, 5]; y = 3 – expected output: 1

   Say for each test case whether it triggers a failure. Justify your answers. [2 marks]

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

Which state is the first error state? [1 mark]

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

| State 0: x = [2, 3, 5] y = 2 i: undefined PC: findLast(...) | State 1: x = [2, 3, 5] y = 2 i: undefined PC: before i=x.length-1; | State 2: x = [2, 3, 5] y = 2 i = 2 PC: after i=x.length-1; | State 3: x = [2, 3, 5] y = 2 i = 2 PC: i>0; | State 4: x = [2, 3, 5] y = 2 i = 2 PC: if (x[i] == y); | State 5: x = [2, 3, 5] y = 2 i = 1 PC: i--; |
| State 6: x = [2, 3, 5] y = 2 i = 1 PC: i > 0; | State 7: x = [2, 3, 5] y = 2 i = 1 PC: if (x[i] == y); | State 8: x = [2, 3, 5] y = 2 i = 0 PC: i--; | State 9: x = [2, 3, 5] y = 2 i = 0 PC: i > 0; | State 10: x = [2, 3, 5] y = 2 i = 0 (undefined) PC: return -1; |

Note: PC stands for ‘Program Counter’