“Software Testing” -- exam prep‘s --

1 Basics

1.1 Define the following terms: equivalence relation, partitioning, graph. Explain the relation between the three terms by an example.

1.2 Equivalence relations and graphs play important roles for the systematic testing of software. Give and briefly explain at least 6 examples.

1.3 What is an Euler graph?
a) Which properties have to be valid for an undirected/directed Euler graph?
b) Design a pseudocode algorithm to determine an Euler path.
c) For which test methods may this algorithm be of help? Give as many as possible.

1.4 Testing versus verification - what are the distinguishing pros and cons?

1.5 Why is an exhaustive testing - in the general case - impossible? Give at least five reasons.

1.6 Explain the terms: test data, test case, test suite, test oracle.

1.7 Give a classification scheme (as a tree) of popular test methods. There should be at least 10 nodes.

2 Static testing

2.1 What is the crucial difference between static analysis and static testing?

2.2 What are the three golden rules of static testing, i.e. the most important rules, which have to be taken into account while performing static testing?

2.3 Why should we apply static testing; or to put it differently: when do human skills overtake computer skills? Find at least 10 arguments.

2.4 Which versions of static testing do you know, and what are their basic differences?

2.5 Design a checklist with your top ten favourite flaws, which can not be discovered by a compiler or static analyser.

3 Test strategies

3.1 Control flow graphs are a very popular modelling technique for white box testing. Why should they better be called control structure graphs?
Note: in the following we use the term „control flow graph” (to stay consistent with the testing community).

3.2 What is the basic assumption for all test methods working on the control flow graph? The same assumption applies for any static analysis technique. Explain the problem by means of an example.

3.3 Explain by means of an example the difference between node coverage (C0) and branch coverage (C1).

3.4 Which programming language constructs have to be used for programming to achieve a difference between node coverage (C0) and branch coverage (C1)? Hint: there are

three!

3.5 Which language constructs in state-of-the-art programming languages can be considered to be structured goto‘s (jumps)?

3.6 Explain Minimal Multiple Condition Coverage and Modified Condition/Decision Coverage (MC/DC); what are the differences?

3.7 Explain the basic ideas of data-flow-based testing and the related defs/uses techniques.

3.8 Given is the following pseudocode program, which examines a (m,n) matrix of character values to determine the number of lines made solely of the character “X“.

```
procedure countXLines (in int m, n; in char[,] matrix) int:
    int count := 0;
    rows: for i from 1 to m do
        columns: for j from 1 to n do
            if m[i,j] /= “X“
            then continue rows
            endif
        endfor columns;
        count++
    endfor rows;
    assert 0 <= count & count <= m;
    return count
endprocedure countXLines;
```

Give its control flow graph in standard notation or as Petri net.

3.9 One approach to compute path coverage takes as denominator the Number of structurally possible Acyclic Paths (NAP). Design an abstract grammar for a syntax-driven NAP computation.

3.10 Consider the program given in 3.8. Determine the Number of structurally possible Acyclic Paths (NAP) for the two cases:
- every loop is entered at most once,
- every loop is entered at most twice.
Explain your answer by use of a syntax tree.

3.11 Consider the program given in 3.8. Determine test cases according to the most popular black box and white box testing strategies.

3.12 Consider the program given in 3.8. Design a cause-effect graph and determine test cases by means of minimal cut sets (minimal T-invariants). Give all intermediate steps.

4 Object-oriented testing

4.1 What are the special challenges of systematic testing of object-oriented software?

4.2 Why does procedure testing differ from method testing, and
why does module testing differ from class testing?

4.3 What are the basic differences of an object-oriented test case compared with a traditional test case?

4.4 Object-oriented testing differentiates between four modality classes. Explain the differences and give an example for each modality!

4.5 Testing of class hierarchies, how do you proceed? Design a guide with the 10 most

important rules.

5 Testing of concurrency

5.1 What are the special challenges of systematic testing of concurrent software?

5.2 Consider the following Java program (https://en.wikipedia.org/wiki/Java_Pathfinder):

```
public class Racer implements Runnable {
     int d = 42;

     public void run () {
          doSomething(1001);
          d = 0;                                  // (1)
     }

     public static void main (String[] args){
          Racer racer = new Racer();
          Thread t = new Thread(racer);
          t.start();

          doSomething(1000);
          int c = 420 / racer.d;                  // (2)
          System.out.println(c);
     }

     static void doSomething (int n) {
          try { Thread.sleep(n); } catch (InterruptedException ix) {}
     }
}
```

a) What is the problem?
b) Design a Petri net to explain the problem.
c) Which testing strategy do you recommend to generally identify this kind of problems?

5.3 Sketch the basic ideas of test tools dedicated to systematic testing of concurrency.

6 Test management

One core issue in test management is to decide how long to keep testing. One option is to stop testing if the number of remaining errors drops below a certain threshold.

Estimate N, the number of remaining errors, for the following scenarios. Explain your solution, i.e. provide all intermediate calculation steps.

6.1 There are 30 artificial errors, which are injected into the software. During the following test phase, 10 of these artificial errors and 7 further (natural) errors are discovered.

6.2 Two completely independent test teams are testing concurrently a given piece of software for a certain period of time. One team finds 10 errors, the other team finds 15 errors. There are 5 errors, which are found by both teams.