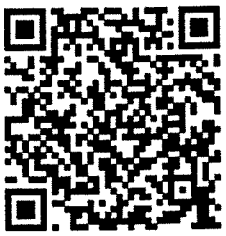
Försättsblad till skriftlig tentamen vid Linköpings universitet



Datum för tentamen	2016-08-17
Sal (1)	TER2
Tid	8-12
Kurskod	TDDD04
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programvarutestning Skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	10
Jour/Kursansvarig Ange vem som besöker salen	Ola Leifler
Telefon under skrivtiden	070-1739387
Besöker salen ca klockan	09:00
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, Email: anna.grabska.eklund@liu.se Phone: +46 13 282362
Tillåtna hjälpmedel	Dictionary/ordbok (printed, NOT electronic)
Övrigt	
Antal exemplar i påsen	

LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Ola Leifler

Written exam

TDDD04 Software Testing

2016-08-17

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ola Leifler, tel. 070-1739387

Instructions and grading

You may answer in Swedish or English.

Your grade will depend on the total points you score on the exam. This is the grading scale:

Grade	3	4	5
Points required	50%	67%	83%

Important information: how your answers are assessed

Many questions indicate how your answers will be assessed. This is to provide some guidance on how to answer each question. Regardless of this it is important that you answer each question completely and correctly.

Several questions ask you to define test cases. In some cases you are asked to provide a minimal set of test cases. This means that you can't remove a single test case from the ones you list and still meet the requirements of the question. Points will be deducted if your set of test cases is not minimal. (Note that "minimal" is not the same as "smallest number"; even when it would be possible to satisfy requirements with a single test case, a set of two or three could still be minimal.)

You may find it necessary to make assumptions in order to solve some problems. In fact, your ability to recognize and adequately handle situations where assumptions are necessary (e.g. requirements are incomplete or unclear) will be assessed as part of the exam. If you make assumptions, ensure that you satisfy the following requirements:

- You have documented your assumptions clearly.
- You have explained (briefly) why it was necessary to make the assumption.

Whenever you make an assumption, stay as true to the original problem as possible.

You don't need to be verbose to get full points. A compact answer that hits all the important points is just as good – or better – than one that is long and wordy. Compact answers also happen to be quicker to write (and grade) than long ones.

Please double-check that you answer the entire question. In particular, if you don't give a justification or example when asked for one, a significant number of points will always be deducted.

1. Black-box testing (4p)

Explain the difference between decision-table testing and equivalence class testing, and give an example where one is applicable and the other is not. (4p)

2. Coverage criteria (8p)

a) Do a partial or complete ordering of the following coverage criteria with respect to their requirements on test cases in ascending order:

   1. Statement coverage
   2. Condition Coverage
   3. Branch Coverage

   (2p)

b) Coverage criteria can be applied to other software artifacts than code. Give an example of such a coverage criterion. (2p)

c) Even when applying the most demanding coverage criterion for software tests, we would need additional information to determine the quality of a test suite. Explain why, by providing a definition of test suite quality. (4p)

3. Unit test automation (6p)

Explain how software tests written using test automation framework such as JUnit/CPPUnit

a) are *executed in a test suite*. Provide an example and explain the difference between running a test suite and running an application. (2p)

b) can *isolate faults* of an application. Describe how fault are isolated to single methods can be detected, as well as faults that are found in the interaction between multiple software components. (2p)

c) can ensure deterministic test behavior. Explain confounding factors that may affect test case results, and how test automation frameworks can eliminate some of those factors. (2p)

4. True/False(6p)

Answer true or false:

a) One goal of software testing is to verify that the system under test (SUT) contains no errors.

b) MM-Paths can be used for integration testing and system-level testing.

c) A bug is the observable effect of executing a fault.

d) Define-use-kill data-flow patterns may not be used for static program analysis.

e) Decision-table testing subsumes Model-based testing.

f) You can automate exploratory testing.

(It's not worth guessing: you get 1p for correct answer, 0p for no answer, and -1p for incorrect answer; you can get negative points on this question.)

5. Black-box testing (16p)

Example from the Waterloo local programming contest, 2006. Author: Piotr Rudnicki.

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

There is an `l`-meter-long ferry that crosses the river. A car may arrive at either river bank to be transported by the ferry to the opposite bank. The ferry travels continuously back and forth between the banks so long as it is carrying a car or there is at least one car waiting at either bank. Whenever the ferry arrives at one of the banks, it unloads its cargo and loads up cars that are waiting to cross as long as they fit on its deck. The cars are loaded in the order of their arrival and the ferry's deck only accommodates one lane of cars. To provide information to passengers waiting for the ferry, the Transportation Authority has requested an application that can list the total queue time for waiting cars.

The input should first consist of two integers $1 \leq l \leq 500$ and $1 \leq m \leq 10000$. Additionally, there should be `m` objects in sequence as input that represent the cars that arrive in this order to be transported. Each object in the sequence gives the length of a car (an integer number of centimeters between 1 and 100 000, inclusive), and the bank at which the car arrives ("`left`" or "`right`").

For each input, the output shall be the number of times the ferry has to cross the river in order to serve all waiting cars.

Select a suitable black-box test technique to test that the method responsible for the calculation above works as intended. Use this technique to devise test cases.

For full points, you must follow the method and you will only receive a few, or even no, points for test cases that are not clearly created as a result of following the chosen method.

State any assumptions about details you deem necessary to make in order to test a method for calculating the waiting time.

6. Symbolic execution (4p)

Symbolic execution works by using so-called *path constraints*. Explain the concept of path constraints, and how it can be used in the process of generating test cases by symbolic execution.

7. Integration testing (6p)

a) State one advantage thread-based integration testing has over other methods. (2p)

b) In top-down integration testing, how many *drivers* are needed at most? (2p)

c) In bottom-up integration testing, how many *test sessions* are needed at most? (2p)

8. Exploratory testing (6p)

a) Can exploratory testing be automated? Justify your answer. (2p)

b) What is exploratory testing especially suited for, as a test method? (2p)

c) In terms of exploratory testing, what is a *charter* and a *tour*? What is the difference between the two? (2p)

9. Modified condition/decision coverage (10p)

Specify a minimal set of test cases for the following function that result in maximal *modified condition/decision coverage*

```
public TriangleType getType(int side1, int side2, int side3)
            throws InvalidTriangleException {
      TriangleType result = TriangleType.NaT;

      int[] s = new int[3];
      int[] sides = new int[] { side1, side2, side3 };

      for (int i = 0; i < 3; i++) {
            s[i] = sides[i];
      }
      if (s[0] > s[1])
            swap(s, 0, 1);

      if (s[0] > s[2])
            swap(s, 0, 2);

      if (s[1] > s[2])
            swap(s, 1, 2);

      if (s[0] <= 0 || s[2] - s[0] >= s[1]) {
            return result;
      }

      if (s[0] == s[2]) {
```

```
            result = TriangleType.Equilateral;
        } else if (s[0] == s[1] || s[1] == s[2]) {
            result = TriangleType.Isosceles;
        } else {
            result = TriangleType.Scalene;
        }

        return result;
    }
```

10. Model-based testing (4p)

Describe *four* different notations that may be used to create models of software systems for the purpose of employing model-based testing. Also, describe the possible advantages and disadvantages with using such models in testing.