Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2015-06-03
Sal	U1,U3,U4,U6
Tid	8-12
Kurskod	TDDD04
Provkod	TEN1
Kursnamn/benämning	Programvarutestning
Institution	IDA
Antal uppgifter som ingår i tentamen	12
Antal sidor på tentamen (inkl. försättsbladet)	7
Jour/Kursansvarig	Ola Leifler
Telefon under skrivtid	070-1739387
Besöker salen ca kl.	10:00
Kursadministratör (namn + tfnnr + mailadress)	Anna Grabska Eklund
Tillåtna hjälpmedel	Dictionary (printed, NOT electronic)

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

Written exam

TDDD04 Software Testing

2015-06-03

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. The maximum number of points is 86. This is the grading scale:

Grade	3	4	5
Points required	42	55	70

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. Terminology (4p)

Explain what “white-box testing” is. Explain one test case design methodology that can be used for white-box testing. (4p)

2. Coverage criteria (8p)

a) Order the following coverage criteria with respect to their requirements on test cases in ascending order:

   1. Path Coverage
   2. Statement Coverage
   3. Branch Coverage

   (2p)

b) Explain when *multiple condition/decision coverage* equals *statement coverage* (2p)

c) Explain why some coverage criteria cannot be quantified in the same way as e.g. statement coverage, and give an example to justify. (4p)

3. Test automation (6p)

Explain advantages that a test automation framework such as CPPUNIT/JUnit have over each of the options below.

Name one advantage and one disadvantage of creating tests for use of a test automation framework instead of

a) performing exploratory testing (2p)

b) generating test cases through symbolic execution (2p)

c) stepping through a debugger (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 both unit testing and integration testing.

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

d) Define-use-kill data-flow patterns may only 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)

You are to test a home alarm system with the following description:

The alarm is set by entering a four-digit code and pressing the "on" button. When activating the alarm, the code has to be entered in full within five seconds or else the system is reset. If the correct code is entered, then, after a given time period, the alarm is activated. Until the alarm is activated, a red LED will flash to indicate that the alarm is about to be activated.

Use a suitable representation to describe the behaviour of this system so that it can be tested using a black-box testing technique. Derive a set of test cases based on the representation, and use a suitable method and metric to determine the character and minimal number of test cases.

6. Symbolic execution and white-box testing (10p)

Explain how symbolic execution works and how it can be used to generate test cases in the following example. Also, provide a set of test cases to obtain 100% branch coverage.

For full points, explain how symbolic execution works in general, how it applies specifically to test case generation, and whether generated test cases are complete or need additional information to provide useful information. Also, describe how branch coverage can be obtained, and provide a clear description of how your test cases achieve branch coverage.

```
public static double calculateBill(int usage) {
        double bill = 0;
        if (usage > 0) {
                bill = 40;
        }
        if (usage > 100) {
                if (usage <= 200) {
                        bill += (usage - 100) * 0.5;
                } else {
                        bill += 50 + (usage - 200) * 0.1;
                        if (bill >= 100) {
                                bill *= 0.9;
                        }
                }
        }
        return bill;
}
```

7. Model-based testing (6p)

Explain the workflow involved in model-based testing with state chart diagrams compared to script-based testing. Describe what may and may not be automated with respect to test design and execution.

8. 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)

9. Exploratory testing (6p)

a) Explain the difference between exploratory testing and ad-hoc testing. (2p)

b) How can exploratory testing be justified as a test method? (2p)

c) In terms of exploratory testing, what is a *tour*? (2p)

10. Modified condition/decision coverage (10p)

Specify a minimal set of test cases for the following function that result in 100% *modified condition/decision coverage*.

```
int rules(int a, int b, int c) {
  if (a < 3 || b > 0) {
    if (b < 1 && c > 2) {
      return b+c;
    }
    return a+c;
  } else if (c > 3 || a > 2) {
    return a-c;
  }
  return a;
}
```

11. Testing case selection (4p)

Explain how to determine the quality of a test suite, by reasoning about how to evaluate different kinds of qualities of different kinds of software products. For full points, you need to reason about how coverage metrics may or may not be used to determine test suite quality.

12. Test automation: True/False (4p)

Answer the following questions true or false:

a) The most commonly automated steps of a software testing workflow are test execution and visualization.

b) Continuous Integration reveals *more types of errors* during integration testing.

c) Continuous Integration reveals *errors earlier* during integration testing.

d) A failing automated test equals an error in the program under test.

(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.)