Information page for written examinations at Linköping University

Examination date	2015-10-21
Room (1)	TER2
Time	14-18
Course code	TDDD04
Exam code	TEN1
Course name **Exam name**	Software Testing (Programvarutestning) Written examination (Skriftlig tentamen)
Department	IDA
Number of questions in the examination	10
Teacher responsible/contact person during the exam time	Ola Leifler
Contact number during the exam time	070-1739387
Visit to the examination room approximately	15:30
Name and contact details to the course administrator (name + phone nr + mail)	Anna Grabska Eklund, Email: anna.grabska.eklund@liu.se Phone: +46 13 282362
Equipment permitted	Dictionary (printed, NOT electronic)
Other important information	
Number of exams in the bag	

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

Written exam

TDDD04 Software Testing

2015-10-21

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. Terminology (4p)

Explain what "black-box testing" is. Explain one test case design methodology that can be used for black-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. Condition Coverage
2. Path Coverage
3. Modified Condition/Decision Coverage

(2p)

b) Does a coverage criterion determine the quality of a single test case or a set of test cases? Justify your answer. (2p)

c) There are two main types of faults that may be detected during software testing: *faults of omission* and *faults of comission*. What does higher code coverage help us detect? Faults of omission, faults of comission or both? Justify. (4p)

3. Unit test automation (6p)

Why should you use a test automation framework such as CPPUNIT/JUnit for *unit-level testing* instead of each of the options below?

Name one advantage and one disadvantage of using a test automation framework for *unit testing* instead of

a) proving low-level properties using static program analysis (2p)

b) only running system-level tests (2p)

c) running an application and inspecting the results (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)

In the USA, the MPAA has defined a number of film ratings. A program is designed to take the age of a person and return the ratings allowed for that person (upper bounds are not inclusive; lower bounds are inclusive, so e.g. "0-13" means up to but not including 13.

Age	Parent present	Ratings allowed
0-13	No	G
0-13	Yes	G, PG
13-17	No	G, PG, PG-13
13-17	Yes	G, PG, PG-13, R
17-	Yes or No	G, PG, PG-13, R, NC-17

Create a minimal set of test cases for checking admittance according to the film rating system. Select an appropriate method for creating test cases, and create a minimal set of test cases based on the method that you choose.

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. 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) 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)

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*.

```
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 > 0) {
      return a-c;
    }
    return a;
  }
```

10. 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.