Information page for written examinations at Linköping University

Examination date	2014-10-22
Room (1)	TER1
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

2014-08-20

Permissible aids

Dictionary (printed, NOT electronic)

Teacher on duty

Ola Leifler, tel. 070-1739387

Instructions and grading

You may answer in Swedish or English.

The total number of points is 81. 40 will be required for pass (3), 56 for grade 4 and 65 for grade 5.

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.

Some questions require you to provide code examples. Your examples do not have to be written in any particular language, or be syntactically correct for full points, but they will have to be unambiguous with respect to the purpose of the question.

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)

Enumerate and describe two general white-box testing techniques, and two black-box testing techniques.

2. Code coverage (8p)

a) Explain why path coverage is insufficient to determine the quality of tests. Give an example of a method that is insufficiently tested even with 100% path coverage. (4p)

b) Give examples of typical software components in object-oriented programs where 100% statement coverage does not add significant information about the behavior of the program. (2p)

c) Explain when 100% statement coverage is not possible. (2p)

3. Test automation (6p)

True or false? Correct answers give 1p, incorrect answers -1p. xUnit-type frameworks include CPPUnit and JUnit, among others that provide similar functionality.

a) A failed automated test case means that the application does not work.

b) xUnit-type tests run in a predetermined sequence, identical between test runs.

c) xUnit-type test frameworks simplify setup of test environments for each test case.

d) Xunit-type test frameworks rely on exception handling for detecting failing tests.

e) xUnit-type frameworks cannot distinguish between failing assertions and errors in test execution.

f) xUnit-type test frameworks can be used for exploratory testing.

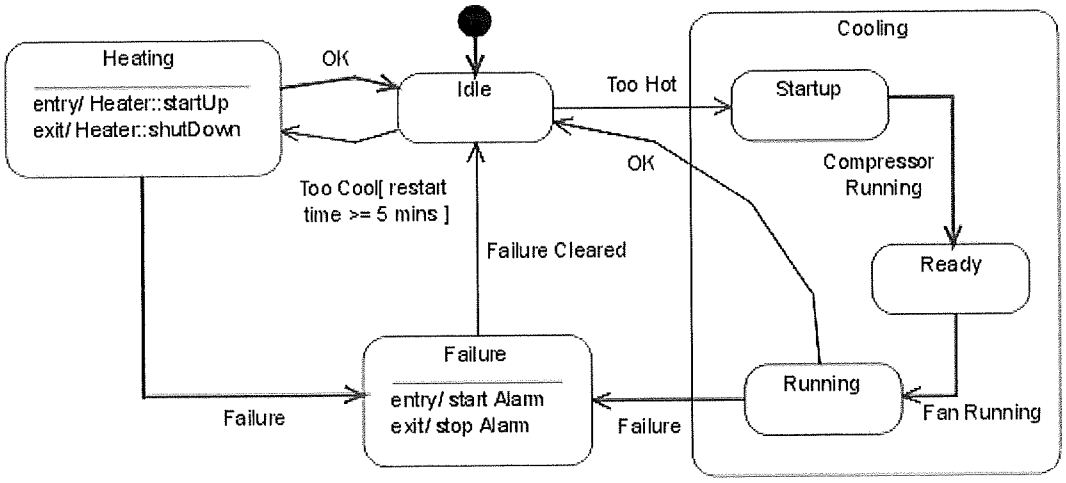
4. Black-box testing (16p)

Assume that you have a 2D strategy game that shall determine on which map tile a user has clicked. There is a method `Tile getTile(int x, int y)` that is supposed to map user input to correct tiles. Set up a scenario for assessing the behaviour of this method, and select an appropriate test method for generating test cases. For full points, you need to describe:

1. A scenario that captures central points of interest for the method. Describe the map used.
2. A suitable test method.
3. A metric used to evaluate the set of test cases generated by the method.
4. A complete set of test cases according to the selected metric.
5. Any assumptions made about the method, or domain.

5. State transition testing (10p)

Use *state transition testing* to test software that is assumed to implement software for the given state transition diagram (state chart) below. Select a quality criterion you can apply to evaluate your test cases. List the test cases that would test the state chart. Also, for full points you should describe possible flaws in the state chart, and possible practical issues with testing the software that implements this state chart.



6. Data-flow testing (6p)

a) Explain how DU-path coverage relates to decision coverage and all-paths coverage. (2p)

b) Explain *define* and *p-use* nodes in a program flow graph using a code example. (2p)

c) Give examples of errors related to data-flow graphs that can be detected at compile-time and at runtime, respectively. (2p)

7. Integration testing (6p)

a) How can *path-based* integration testing be justified instead of *bottom-up/top-down* integration testing? (2p)

b) Give a concrete example of when top-down integration testing would be preferrable to bottom-up integration testing, and vice versa. (2p)

c) In neighborhood integration testing, how do you calculate the number of neighborhoods in a given call graph? (2p)

8. Testing methods (6p)

True or false? Correct answers give 1p, incorrect answers -1p.

a) Software testing is tantamount to writing automated test cases.

b) Exploratory testing is a repeatable process that can provide measurable, quantitative results on software quality.

c) System-level testing cannot make use of threads the way unit and integration testing can.

d) System-level testing only exercises non-functional tests.

e) Test-driven development requires test automation frameworks.

f) Thread testing is only used at the system level.

9. Modified condition/decision coverage (10p)

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

```
/**
 * @param a
 *            0...3
 * @param b
 *            0...3
 * @param c
 *            0...3
 * @param d
 *            0...3
 * @return 1...4
 */
static int f1(int a, int b, int c, int d) {
        if (a % 2 == 1 || b % 2 == 0) { // D1
                if (a > 1 || c != 3) { // D2
                        return 1;
                }
                return 2;
        }
        if (c > 1 && d != 1) { // D3
                return 3;
        } else {
                return 4;
        }
}
```

10. Test method selection (9 p)

What test methods would be a good fit for the following three types of products:

a) Creating simulation software of a set of related regional hydrological systems such as ground saturation, dissipation, and evaporation as separate systems. (3p)

b) Rostering systems that feed information to several different payroll information systems on working hours after a roster period has ended, and collect information on employments and personal information before the rostering period starts. (3p)

c) Merging several legacy financial systems into a general framework, where existing systems are to be integrated as service providers in a financial application framework. (3p)

Explain your assumptions about these systems as you justify your choice of test design methods. Also explain if your choice of test method applies to the unit testing, integration testing, or system testing level, or several of them. Your justifications will have to relate clearly to *specific features* of each product type, and *poorly justified or inappropriate assumptions will lead to a deduction of points*.