Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-06-03
Sal	???
Tid	14-18
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.	15:00
Kursadministratör (namn + tfnnr + mailadress)	Anna Grabska Eklund
Tillåtna hjälpmedel	Inga

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

Written exam

TDDD04 Software Testing

2014-06-03

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. The following grading scale is **preliminary** and the limits may be lowered after grading.

Grade	3	4	5
Points required	35	50	65

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. Briefly explain one test case design methodology that can be used for black-box testing. (4p)

Black-box testing relies on specifications. Test cases may be created using, for example, equivalence classes, boundary values, or decision tables.

2. Coverage criteria (8p)

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

1. Decision Coverage
2. Condition Coverage
3. Branch Coverage

1 and 3 can be considered synonymous, and 2 is parallel to 1 and 3.

(2p)

b) Explain *Modified decision/condition coverage* with a code example (2p)

MCDC requires that every decision value should be tested, as well as all condition values that contribute to those decisions.

```
// x = 3, y = 8 => T, y contributes to decision evaluating to true
// x = 3, y = 7 => F
// x = 2, y = 7 => T, x contributes to decision evaluating to true
// x has assumed values 2,3 to make the condition (x < 3) both true and false.
// y has assumed values 7,8 to make the condition (y > 7) both true and false.
        if ((x < 3) || (y > 7)) {
                System.out.println("One path taken");
        } else {
                System.out.println("Another path taken");
        }
```

c) Explain coverage criteria that can be used for loops (4p)

For simple loops, we can calculate coverage based on boundary value testing on the number of iterations performed. For nested loops 1..N, starting with N where N indicates the number of nested loops, and a currently tested loop j, we can perform a minimum number of iterations of loops 1..j-1, and a typical number of iterations of loops j+1..N and do boundary value testing of loop j.

3. Test automation (6p)

When should you use a test automation framework such as CPPUNIT/JUnit instead of

a) stepping through a debugger (2p)

To automate commonly executed tests, when the result of executing a function is known and the only object of interest. To track down an existing bug, however, it might be insufficient to use a test automation framework depending on the information that can be revealed by the classes under test.

b) writing debug information to the console (2p)

When debug information becomes too verbose, isolated tests can reveal more information than debug log information. However, for longer traces of system behavior and to catch incidents, it might be necessary to write debug information to a console and use it for forensic analysis of system crashes that automated tests have not caught.

c) running an application and inspecting the results (2p)

To ensure that functionality that has previously been tested manually continues to work when functionality is changed, automated tests are useful. When determining how an entire system works, and how users may interact with the system, running the application is probably a more useful method.

4. Easy points (6p)

Answer true or false:

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

b) MM-Paths can be used for both unit testing and integration testing. FALSE

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

d) Requirements reviews are in general better than code inspections at finding faults in applications. FALSE

e) Equivalence-class testing subsumes decision-table testing. FALSE

f) The control script for data-driven test scripts requires little effort to set up. FALSE

(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)

The Swedish Tax Authority Skatteverket has the following rule for paying road user charges (tolls) for foreign heavy vehicles, simplified for this question:

“[The toll] depends on what distance you are intending to drive on a toll road, the emission category of your vehicle (EURO 0, 1 or 2) and the number of axles of the truck or truck/trailer combination. You can pay for day [or week]. For more details, please see the [tariff table below].

Note that the chargeable period for day is 00.00-24.00. So, if a journey on a toll road begins, for example, at 21.30 on one day and finishes at 02.30 next day, payment for two days is required.

Tolls are paid in SEK.”

Toll Tariff 2014

Max axles	**3**	**3**	**3**	**4**	**4**	**4**
Emission Class	**0**	**1**	**2 or cleaner**	**0**	**1**	**2 or cleaner**
1 day	67	67	67	67	67	67
1 week	220	194	169	347	313	279

For a hypothetical software system that produces the toll charge given the information above, select a test case design methodology and use it to define a minimal set of test cases (according to that methodology). Justify your choice of methodology! You will be evaluated on both your choice of methodology, justification, and application of the methodology.

Given the definitions in the text, using decision tables would probably be a good idea. Some of the conditions are clearly separable in equivalence classes (*max axles*, *emission class*) whereas the time would contribute according to some formula that gives us a toll to pay based on the duration and/or distance we travel. To understand how to apply the decision table method on this problem, we could first dissect the relationship between daily and weekly costs for each toll class:

Cost	Days	Weekly toll cheaper for
67	1	None
134	2	None
201	3	C1: 3, C4: 1,2+
268	4	C1: 3, C4: 0
335	5	C1: 4, C4: 1,2+
402	6	C1: 4, C4: 0

It is stated that the duration should be calculated with no consideration to the exact times of entry and exit, only the dates. This is reflected in our decision rules below. Dates are denoted d1, … d8 and represent arbitrary dates with an index to represent time difference between them. When entering and leaving the same day, it is still counted as one day. We choose arbitrary attribute values for 1 and 2 days, and choose values for those classes that we assume should pay a weekly toll for subsequent time intervals.

Rule												
C1: Axles	3	4	3	4	3	3	4	3	4	4	4	4
C2: Start Time	d1	d1	d1	d1	d1	d1	d1	d1	d1	d1	d1	d1
C3: End Time	d1	d2	d3	d4	d4	d4	d5	d5	d6	d6	d6	d7
C4: Emission Class	0	1	2+	2+	1	2+	1	0	0	1	2+	0
D: Toll	67	67	134	201	194	169	268	220	335	313	279	347

Based on these rules, we create test cases:

Test	1	2	3	4	5	6	7	8	9	10	11	12
C1: Axles	3	4	3	4	3	3	4	3	4	4	4	4
C2: Start Time	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00	1/6 12:00
C3: End Time	1/6 18:15	2/6 00:00	3/6 10:27	4/6 12:10	4/6 12:00	4/6 00:00	5/6 10:15	5/6 13:00	6/6 00:00	6/6 12:01	6/6 13:00	7/6 00:00
C4: Emiss	0	1	2	3	1	2	1	0	0	1	3	0

ion Class												
D: Toll	67	67	134	201	194	169	268	220	335	313	279	347

6. Basis path testing (10p)

Use *basis path testing* to define test cases for the following programs.

```
int f(int x, int y) {
  bool x_neg;
  if (x > 0) {
    x_neg = false;
  }
  else {
    x_neg = true;
    x = -x;
  }

  if (y > 0) {
    if (x_neg) {
      y = -(y * y);
    }
    else {
      y = y * y;
    }
  }
  else {
    y = -(y * y);
  }
  return x + y;
}
```

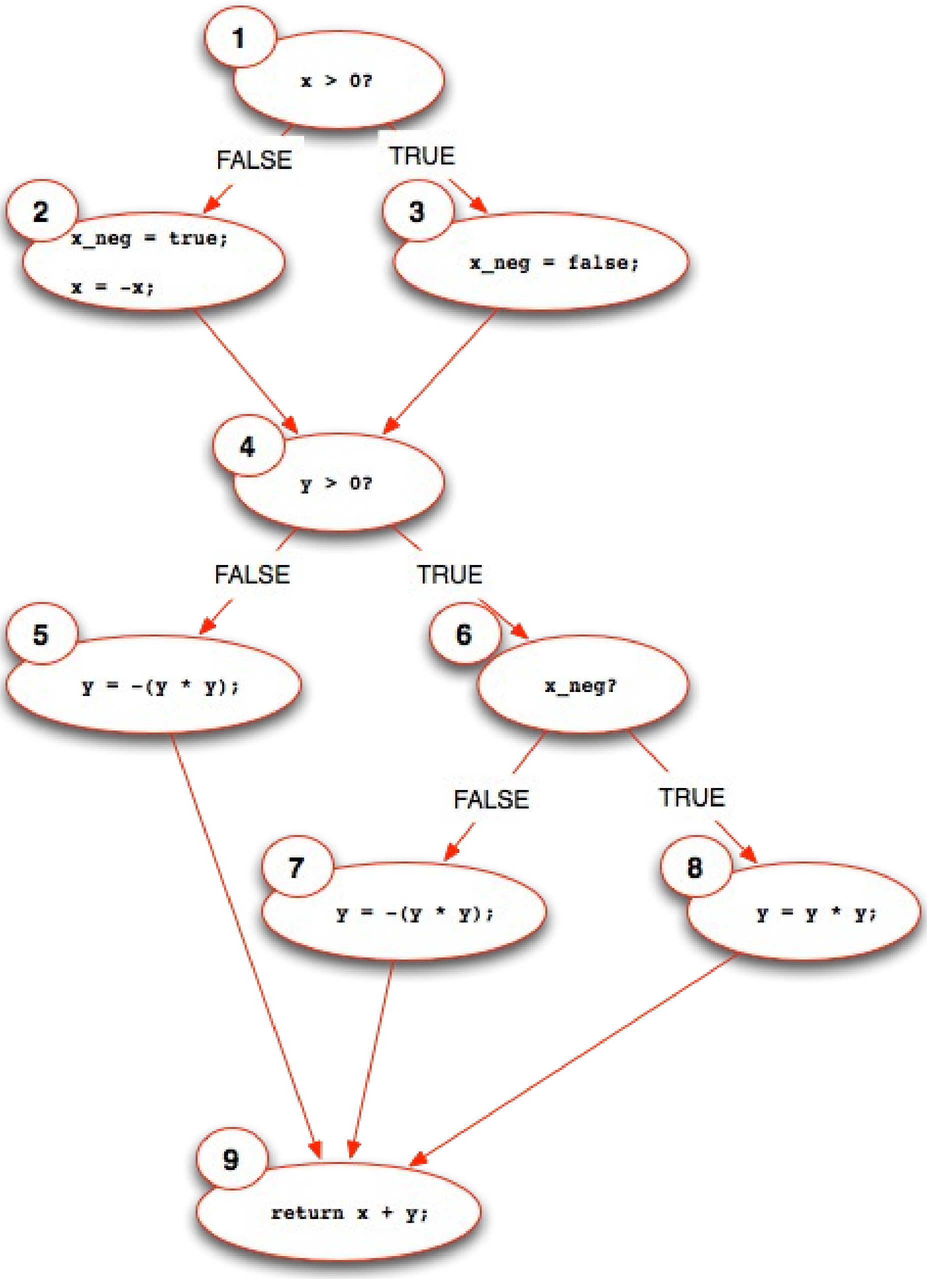
You are required to draw the appropriate graphical representation of the program and illustrate the process of selecting paths. For each test case, indicate which basis path the test case corresponds to, in addition to the other information required for a test case. You will be assessed on the quality and completeness of your test cases, as well as your explanation of the basis path selection process.

Given the graph below, assume that we start with path 1,2,4,5,9 (A). We change the decision outcome in 1 and obtain 1,3,4,5,9 (B). We change the decision outcome in 4 and obtain 1,3,4,6,7,9 (C). Last, we change the outcome in 6 and obtain 1,2,4,6,8,9 (D) since `x_neg` is only set to true if we choose node 2.

To test A,B,C,D, we need value of x > 0 and x <= 0, and values of y > 0 and y <= 0.

Path	X	Y	Expected result:
A	0	0	0
B	3	0	3

C	1	4	-15
D	0	1	1


1
x > 0?
FALSE
TRUE
2
x_neg = true;
x = -x;
3
x_neg = false;
4
y > 0?
FALSE
TRUE
5
y = -(y * y);
6
x_neg?
FALSE
TRUE
7
y = -(y * y);
8
y = y * y;
9
return x + y;

7. Code Complexity measures (6p)

Explain *cyclomatic complexity* and *essential complexity*, and their relationship to testing. For full points, provide a code example that describes the relationship between *cyclomatic complexity* and *essential complexity*.

Cyclomatic complexity (CC) determines the size of the set of basis paths and thus test cases needed for control-flow-based testing. Binary decisions + 1 equals CC. Essential complexity reduces all well-structured decision points to single nodes. Unstructured programs have an essential complexity greater than 1.

8. Integration testing (6p)

a) State one advantage big-bang integration has over most other methods. (2p)

No stubs or drivers are required, and errors are not concealed by fake implementations.

b) In the context of integration testing, what is a *driver*? (2p)

A simplified substitute for a component that, at runtime requires and makes use of the currently tested component. Typically used in bottom-up integration testing.

c) In the context of integration testing, what is a *stub*? (2p)

A simplified substitute for a component that is used by the currently tested component. Typically used in top-down integration testing.

9. Exploratory testing (6p)

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

Exploratory testing is goal-directed and explores different options for obtaining a specific goal, or performing a specific task as a user. Ad-hoc testing is not directed by such a goal or task.

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

It can be more likely to detect errors that go undetected by scripted tests that do not attempt to probe the system for unexpected behavior.

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

Defines the mission of the current exploration of the system.

10. Multiple condition coverage (10p)

Specify a minimal set of test cases for the following function that result in 100% multiple condition coverage.

```
int rules(bool a, bool b, bool c, bool d) {
  if (a || b) {
    if (~b || c) {
      return 1;
    }
    return 2;
  }
  else if (c && d) {
    return 3;
  }
  return 4;
}
```

```
All combinations of conditions (a b), (b c), (c d) must assume
values T & F to achieve 100% MCC. Thus:

1. a = F, b = F, c = F, d = F, expected result: 4
2. a = F, b = F, c = F, d = T, expected result: 4
3. a = F, b = F, c = T, d = F, expected result: 4
4. a = F, b = F, c = T, d = T, expected result: 3
5. a = F, b = T, c = F, d = ?, expected result: 2
6. a = T, b = T, c = T, d = ?, expected result: 1
7. a = T, b = F, c = F, d = ?, expected result: 1
8. a = T, b = F, c = T, d = ?, expected result: 1

Each combination of values of the three pairs must be present in
the test cases. This table shows why these 8 test cases are enough
instead of 16 test cases, since the value of "d" is irrelevant in
half of them.
```

	(a || b)	(~b || c)	(c && d)
FF	1,2,3,4	5	1
FT	5	6	2
TF	7,8	7	3
TT	6	8	4

11. Testing in agile development (4p)

Describe two techniques for system-level testing in agile development, where fast releases and continuous delivery is critical. For full points, use concrete examples of how to implement your chosen techniques.

To ensure fast releases, the system-level testing can to be performed incrementally with in-house exploratory test methods for new functionality, and incremental rollouts of new features for selected groups of users, either internal user groups or randomized subsets of the user base.

12. More easy points (4p)

Answer the following questions true or false:

a) Automated testing is effective for ensuring that old bugs are not reintroduced. TRUE

b) Automated tests are cheap to create. FALSE

c) Automated testing is appropriate in an agile environment. TRUE

d) By automated testing people usually mean testing where tests are run automatically, but inputs and outputs are created by hand. TRUE

(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.)