Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-08-20
Sal	TER2
Tid	8-12
Kurskod	TDDD04
Provkod	TEN1
Kursnamn/benämning	Programvarutestning
Institution	IDA
Antal uppgifter som ingår i tentamen	10
Antal sidor på tentamen (inkl. försättsbladet)	7
Jour/Kursansvarig	Ola Leifler
Telefon under skrivtid	070-1739387
Besöker salen ca kl.	09:00
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

2014-08-20

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)

Explain the concepts of, and relationships between, an *error*, a *fault*, a *failure* and an *incident* in the context of software testing, and give examples of each.

2. Code coverage (8p)

a) Describe the relationships between the following code coverage metrics: *decision coverage*, *condition coverage*, and *modified condition/decision coverage*. Use a code example to justify your answer (4p).

b) Explain when decision coverage is better than statement coverage. Use a code example to justify your answer. (2p)

c) Explain when 100% path coverage is not possible. Use a code example to justify your answer. (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) xUnit-type test frameworks provide an API for writing tests.

b) xUnit-type test frameworks provide a runtime environments for tests.

c) Automated test frameworks generate test cases for an application.

d) xUnit-type test frameworks are only used for unit tests.

e) Automated test frameworks can be used for acceptance testing.

f) xUnit-type test frameworks rely on exception handling for asserting correct program behavior.

4. Black-box testing (16p)

The Swedish Social Security Office (Försäkringskassan) provides housing allowances (bostadsbidrag) to individuals and households based on some selection criteria. Upcoming revisions of the criteria, and revised testing practices of the software used to process application forms, mean that you are responsible for selecting test cases to ensure correct implementation of the business rules surrounding housing allowances. The software should be tested using techniques that you have learned during the course to obtain a good set of test cases. For full points, you need to:

1. Identify a suitable metric for test cases
2. Identify suitable test cases for the method, including assumptions about how to interpret the specification as given below and how to test the given method.
3. Use an appropriate method for obtaining the test cases.

The specification reads as follows:

If you are between the ages of 18 and 28 and live alone, you can receive housing allowance if your income is less than SEK 86,730/year. If you are married or cohabiting and you are both between the ages of 18 and 28, you can jointly receive housing allowance if you together have an income that is less than SEK 103,720/year.

The current interface specification looks as follows:

```
public interface HousingAllowanceCheck {

        boolean eligibleForAllowance(Person p);

}
```

Make reasonable assumptions about the properties of the class `Person` in order to test the specification given.

5. Basis path testing (10p)

Use *basis path testing* to define test cases for the following Java method:

```
/**
 * Updates the dependencies list of all cells according to the new
 * dependencies list of cell c.
 *
 * @param c
 *            the cell we are working on
 * @param cDeps
 *            the dependencies list of cells that c depends on
 */
private void updateDependencies(Cell c, List<String> cDeps) {
        String cName = c.getName();
        // Iterating over the cells set
        for (String cellName : cells.keySet()) {
                // Dependencies list of the current cell
                List<String> l = dependencies.get(cells.get(cellName));
                // 1. If we know that the c depends on the current cell...
                if (cDeps.contains(cellName)) {
                        // ... and its list doesn't contain c ...
                        if (!l.contains(cName))
                                // .. then we add c to its dependencies list
                                l.add(cName);
                }
                // 2. Or, if we know that c doesn't depend on the current cell..
                else {
                        // .. and its list contains c..
                        if (l.contains(cName))
                        // .. then we remove c from its dependencies list.
                                l.remove(cName);
                }
        }
}
```

You are required to draw the appropriate graphical representation of the program and outline the basis paths in the graph. Devise test cases for each path and for each test case, indicate which basis path the test case corresponds to, in addition to the other information required for a test case.

The fields `cells` and `dependencies` are fields of the class the method is defined in. State any assumptions about *how* you intend to execute the test, although you will not have to write code examples. You will be assessed on the quality and completeness of your test cases, including *how to test the given method*, as well as your explanation of the basis path selection process.

6. Data-flow testing (6p)

a) Define the concept of DU-paths in terms of the program graph G of a program P (denoted G(P)) where nodes correspond to program statements and expressions. (2p)

b) Give an example of a definition-clear DU-path and explain what a definition-clear du-path is. (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) Explain *path-based* integration testing and how it can be justified instead of *top-down*/*bottom-up* integration testing. (3p)

b) Use an example to describe the process of MM-path selection. (3p)

8. Testing methods (6p)

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

a) Exploratory testing requires more domain knowledge than script-driven testing.

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

c) Model-based testing means that you test a model of the software, not an implemented software system.

d) There are model-based testing tools that generate test cases from models of user behavior.

e) Test-driven development requires test automation frameworks.

f) Code inspections are less effective than acceptance tests at finding faults.

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 < 3 || b > 2) {
		if (a > 2 || c != 3) {
			return 1;
		}
		return 2;
	}
	if (c % 3 == 0 && d != 1) {
		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) Massive multiplayer online role-playing game, continuously improved and released to active as well as new players over a number of years. (3p)

b) Firmware for controlling battery backups and failover power supplies at a nuclear power plant, to be incorporated with drivers for use in larger control systems for the power plant. (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*.