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

Written exam

TDDD04 Software Testing

2017-01-05

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 (6p)

Describe and explain the relationships between the following terms:

- Error
- Defect
- Failure
- Incident
- Test case
- Test

2. Coverage criteria (8p)

a) Is it possible for a method with *multiple return statements* to be tested with 100% decision coverage using a *single test case*? Justify your answer. (2p)

b) Give examples of two software artifacts that need to be subjected to testing, but for which coverage criteria cannot be defined in a similar manner as for code. (2p)

c) Explain when it may be inappropriate to target 100% statement coverage when writing test cases. (2p)

d) Conduct a partial ordering of the following coverage criteria, in ascending order of requirements on test cases (2p):

- Condition coverage
- Decision coverage
- Path coverage
- Statement coverage

3. Defect classification (6p)

You are given the following description of a software defect:

> "*Entered by user*: When I enter values quickly in our Results Manager module of of LADOK 2.0, and I press the Submit button to enter results for a large exam with more than 250 students, I am thrown out of the system and have to start all over!
>
> *Entered by technician*: The security requirement for session inactivity timeout is set to 15 minutes in LADOK 2.0, and in the system design, activity has been interpreted as pressing buttons such as Submit. Entering values in a form should also have been considered as activity so users are not thrown out when entering results."

Fill in appropriate values for the columns in the table below (copy to a separate paper first) when you classify the defect above.

Fault/Defect	Attribute	Value
	Asset	
	Artifact	
	Effect	
	Mode	
	Severity	

4. True/False(6p)

Answer true or false:

a) A system under test exercises test cases.

b) You can measure code coverage of test cases as well as only the system under test.

c) Mutation testing mutates tests to find the best tests for a piece of code.

d) With xUnit type test frameworks, you cannot assert that a method throws an expected exception, only that it returns expected values.

e) Inspections are more effective at finding defects in design documents than structural testing.

f) You can automate exploratory testing by using Model-based testing techniques.

You get 1p for correct answer, 0p for no answer, and -1p for incorrect answer. However, you can not get negative points for this question.

5. Black-box testing (16p)

At your company Pollution Detectors Inc, you have developed a product that measures the levels of microscopic particulates in the air and displays a warning on a public display if the pollution level exceeds a given threshold. Here are the specifications of the threshold values for two types of such particulates:

Name	Size	Threshold (micrograms/m^3)	Time frame
Fine particulates (PM2.5)	2.5 mikron	10	Annual mean
		25	24h mean
Coarse particulates (PM10)	10 mikron	20	Annual mean
		50	24h mean

For example, for PM2.5, annual mean values may not exceed 10 micrograms per cubic meter, and at any 24h interval, the mean value may not exceed 25 micrograms per cubic meter.

There is a method for adding measurement values to the system

```
void addPollutionValue(Measurement m)
```

This method is used to update the system with new `Measurement` values, with the time of invocation as the given time. A `Measurement` describes the particulate type (PM2.5 or PM10) and the currently measured averaged value in an area, in mikrograms per cubic meter. A `Measurement` object thus has an attribute `pType` (enum value, `{Fine, Coarse}`) and an attribute `value` (float).

The graphical display regularly asks for warning values, using the method

```
Set<Status> getCurrentStatus()
```

`Status` is an enum value `{OK, PM25Annual, PM25Daily, PM10Annual, PM10Daily}`, where the values indicate either pollution levels below the thresholds (`OK`), or above the thresholds for either PM2.5 or PM10, for annual or 24h mean values, respectively. That is, the method may indicate that several of the thresholds are violated at the same time.

Follow a suitable test case design method in order to test the given methods. Justify your choice of test case design method, and any assumptions you make about the system. The system should be tested under reasonable realistic operating conditions. You should describe how you plan to initialize the system for the test cases that you believe appropriate.

You can receive up to 8 points for an appropriate representation of the problem, and up to 8 points for an appropriate translation of your representation into test cases. An excessive amount of test cases (based on the choice of test case design method) will lead to a deduction of points, as well as an incomplete set of test cases.

6. White-box testing (16p)

You are to test a method for calculating the greatest common denominator between two numbers a and b.

Given the code example below,

1. create a set of basis paths for the code, and justify why your set is *sufficient*, (6p)
2. create test cases for 100% *decision coverage* based on the basis paths (6p), choosing values that are interesting to test apart from helping you achieve 100% decision coverage, and
3. explain how Symbolic Execution may be used one the given method to generate possible test cases. Also, explain how *path constraints* work when generating test cases, and state whether the test cases would be complete or require additional code after they have been generated. (4p)

Note that some of the paths will require several iterations of the loop, meaning several executions of decisions. A path corresponding in this case must ensure that the decisions required are taken *at some point during the execution*.

```
/**
 *
 * Calculate the greatest common denominator (GCD) of two numbers a and b
 * using the property that the GCD divides
 * the difference between a and b.
 *
 * @param a
 * @param b
 * @return the GCD of a and b
 */
public static int gcd(int a, int b) {
        if (a == b) {
                return a;
        } else if (a > b) {
                return gcd(a - b, b);
        } else {
                return gcd(a, b - a);
        }
}
```

7. Integration testing (6p)

a) Name the types of additional scaffolding code that may be needed for integration testing. Explain which tradeoffs are made with respect to the ability to *detect* faults, and the ability to *locate* faults when using scaffolding code. (3p)

b) Name and explain two critical success factors in integration testing. (2p)

c) Describe the main contents of an integration testing plan. (1p)

8. Exploratory testing (4p)

Detects faults of omission		
Does not detect faults of omission		
	Low maintenance cost	High maintenance cost

a) Fill out the table above and mark where Exploratory testing (X) and automated testing (A) would fit. Justify your answer (2p).

b) Explain the difference between detecting and locating faults, and whether using exploratory testing is better at detecting or locating faults compared to automated testing. (2p)

9. System-level testing (4p)

1. Describe MM-path testing (1p)
2. Name two types of test case generation strategies (2p)
3. Explain the difference between verification and validation of a system (1p)

10. Model-based testing (3p)

1. Give two examples of formalisms to use for constructing a model of software to be used for testing. (2p)
2. Name a coverage criterion that can be applied in model-based testing. (1p)