Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2019-08-21
Sal	
Tid	
Kurskod	TDDD04
Provkod	
Kursnamn/benämning	Programvarutestning
Institution	IDA
Antal uppgifter som ingår i tentamen	
Antal sidor på tentamen (inkl. försättsbladet)	
Jour/Kursansvarig	Lena Buffoni
Telefon under skrivtid	
Besöker salen ca kl.	
Kursadministratör (namn + tfnnr + mailadress)	Anna Grabska Eklund
Tillåtna hjälpmedel	Ordbok

LiTH, Linköpings tekniska högskola
IDA, Institutionen för datavetenskap
Lena Buffoni

Written exam

TDDD04 Software Testing

2019-08-21

Permissible aids
Dictionary (printed, NOT electronic)

Teacher on duty
Lena Buffoni

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. Integration Testing (14p)

Given the code below:

```
int f1(int a, int b, int c){
int res;
if (a>b)
  res = f2(a, c);
else
  res = f2(b,c);
}

int f2(int a, int b){
int res;
if (a>b)
 res = f3(a);
else
 res = f3(b);
return res;
}

int f3(int a){
 int res;
 res = a*a;
 return  res;
}
```

a. Calculate the set of module execution paths (MEP) (6p)
b. Draw a MEP path graph for the test case where a=1, b=2, c=3 (4p)
c. How many paths are necessary for MEP testing? (1p)
d. Give one advantage of MEP integration testing (1p)
e. Name the types of additional scaffolding code that may be needed for integration testing. (2p)
f.

2. Black Box testing (10p)

a. Explain how pair-wise testing works and give a scenario where it is useful. (3p)
b. The test table below is used to decide if a user is entitled to a phone upgrade. Generate a minimal set of test cases based on this table. (6p)
c. Besides generating test cases, what are test tables useful for? (1p)

Months since last phone upgrade	> 24	12-24	12-24	12-24	<12
Is premium customer	--	yes	no	no	--
Has 20000 bonus points	--	--	yes	no	--
Is eligible for free phone upgrade	yes	yes	yes	no	no

3. White-box testing (12p)

```
bool getParity(unsigned int n)
{
```

```
    bool parity = 0;
    while (n)
    {
        parity = !parity;
        n      = n & (n - 1);
    }
    if (parity) print('pair'); else print ('impair');
    return parity;
}
```

a. For the code above calculate the cyclomatic complexity and provide the set of basis paths. (6p)
b. Based on these basis paths, provide a set of test-cases, trying to choose values meaningfully. (2p)
c. Is the set of basis paths unique? Justify your answer. (2p)
d. Describe how mutation testing works on the code below, by giving an example of two **different** mutations and say whether they are detected by your test suite or not. (4p)

4. Defect classification (10p)

a. You are asked to classify the following report using the table below (copy it out on paper first). (6p)

"Alexandra notices that in the simulation software she is using, if the simulation time is longer than 5000 seconds, the software crashes and does not save the latest changes to the file."

b. Give two ways defect taxonomies can be helpful in testing (2p)
c. Explain what is a risk classification taxonomy and how it is helpful in testing (2p)

Fault/Defect	Attribute	Value
	Asset	
	Artefact	
	Effect	
	Mode	
	Severity	

5. Test Planning (6p)

a. Are the total number of tests and the percentage of tests passed good criteria for evaluating test-suite quality? Justify your answer. (2p)
b. What is robustness testing? (1p)
c. What is smoke testing? When is it helpful? (2p)

d. Give two limitations of test-driven development. (1p)

6. Software Test Automation (8p)

a. Describe three activities in the model-based test-generation process. (3p)
b. Name two qualities that a good model for testing should have and illustrate them with an example (4p)
c. Give one scenario where test automation is not a good strategy. (1p)

7. Model Based Checking (6p)

a. Explain the principle of symbolic testing (2p)
b. Fill in the following table with True/False (4p)

	Exhaustive	**Partial**	**Aims for completeness**	**Aims for soundness**
Model checking				
Symbolic execution				

8. Easy Points – True or False (8p)

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

a. Test automation helps managers track testing progress using test suite size
b. In test-driven development tests are written based on the program structure
c. The goal of software testing is to prove the absence of bugs
d. The defect detection percentage of a test suite is one of the indicators that can be used to evaluate the test suite effectiveness
e. Black-box testing can be used on the integration-testing level
f. Additional test cases for the same equivalence class require more effort but increase test-suite effectiveness at finding faults
g. Exploratory testing is dependent on the quality of the requirements
h. A minimal test set that achieves 100% path coverage will generally detect more faults than one that achieves 100% statement coverage.