Vrije Universiteit, Department of Computer Science

Examination paper for **Software Testing**
28 May 2013 15:15-18:30 KC159

This is a closed book written exam.

No printed material or electronic devices are admitted for use during the exam.

The answers have to be given in English or Dutch.

Both homework and exam are compulsory and graded on a 1 to 10 scale.

The exam grade is calculated as (Q1+Q2+Q3+Q4+Q5 +10)/10.

The final grade is calculated as 0.5*homework + 0.5*exam

A pass is given if both homework and exam components are >= 5.5.

	Q1	**Q2**	**Q3**	**Q4**	**Q5 (code)**	**Σ Qi**	**Maximum credits= (ΣQi+10)/10**
a)	**3**		**3**		**3**		
b)	**5**		**9**		**5**		
c)	**3**				**8**		
d)	**3**				**5**		
e)	**3**				**3**		
f)	**7**						
g)	**6**						
Total	**30**	**12**	**12**	**12**	**24**	**90**	**10**

Good luck!

Q1. Concepts [30p]

a. What is a **latent fault**? Give an example [3p]
b. Define **defect density**. Why is defect density useful in testing? [5p]
c. What is a **test adequacy criterion**? Give an example. [3p]
d. What is the difference between **verification** and **validation**? [3p]
e. Why is it important to link test cases with requirements ? [3p]
f. What means **regression testing**? Enumerate a few regression testing techniques and explain in one sentence how do they work [7p]
g. Safety critical software needs a special kind of testing, regulated by standards. Enumerate a few special procedures that have to be applied when testing safety critical software. [6p]

Solutions

a). A latent fault is a fault that did not produced a failure yet. example :millennium bug
b) Defect density is a metric defined as the ratio of number of faults to the size of the tested software component, usually faults/KLOC . it is useful to be monitored in order to estimate when to stop testing (when defect density drops under a certain value), to determine high-risk components that will require more attention in future projects, is a good metric to compare different testing techniques.
c)A test adequacy criterion can decide if a test suite is good or not. Given a program P and a test suite T, a test adequacy criterion is a predicate that is true (satisfied) or false (not satisfied) of a <P,T> pair. Example: all decision coverage criterion for white box testing.
d) Verification compares the end products of a development phase with the end documents from the previous phase ("Are we building the system right?", while validation compares these products with the user's requirements and wishes ("Are we building the right system?").
e) it is important in the test case design to make a link to the requirements using tags because we need to know for example how many requirements have been already tested. We also want sometimes to trace a failed test case to the requirement it is related to.
f)regression testing checks if the correction of a bug or changes needed to add extra functionality do not affect the quality of the program. The problem is to decide which tests to run again. the options are: test all, modification transversal , test minimization, test prioritization.
g) Here all kind of standard regulations have to be mentioned.
For example some techniques are required such as: inspections, MC/DC coverage (required by DO 178b standard for aviation) or at least 100% decision coverage, combinatorial testing n-way with n>3, formal verification of the design, peer reviews, fault tree analysis, probabilistic testing, all bugs have to be traceable, programmers that make 3 bugs are dismissed, simulations, compiler certification, verify safety-behaviour during degraded and failure conditions, logging, avalanche testing, etc.

Testing from requirements Q2-Q4 [36p]

Q2. [12p]

You have to test the validity of the birthday input in a GUI. The user has to enter his date of birth by filling 3 separate text fields : month (mm), day (dd) and year (yyyy). A year is valid if it is between 1812 and 2013.
Generate test cases you think will appropriately test this Birthday input. The output of each test case should be Valid or Invalid. Explain your strategy.

Solution:
We identify 3 input variables: month, day and year.
Because the valid inputs are ranges we chose for equivalence partitioning (EP).
We add first more requirements based on our calendar knowledge.
month is an integer, 1<=month<=12
and
day is an integer, 1=<day<=31.

We can apply a simple EP.

The valid equivalence classes are:

M1= {month: 1 <= month <= 12}
D1 = {day: 1 <=day<=31}
Y1= {year: 1812 <= year <= 2013}

And the invalid equivalence classes are:

M2 = {month : month < 1}
M3 = {month : month > 12}
D2 = {day : day < 1}
D3 = {day : day > 31}
Y2 = {year: year < 1812}
Y3 = {year : year > 2013}

Test case specification:
We can make the single fault assumption and generate this set of test cases:
TC1. All variables are in valid classes: Output: valid
TC2-TC3. Month is in invalid class, Day and Year are valid: Output: invalid
TC4-TC5. Day is invalid, Month and Year are valid: output: invalid
TC6-TC7: Year is invalid, month and day are valid, Output: invalid

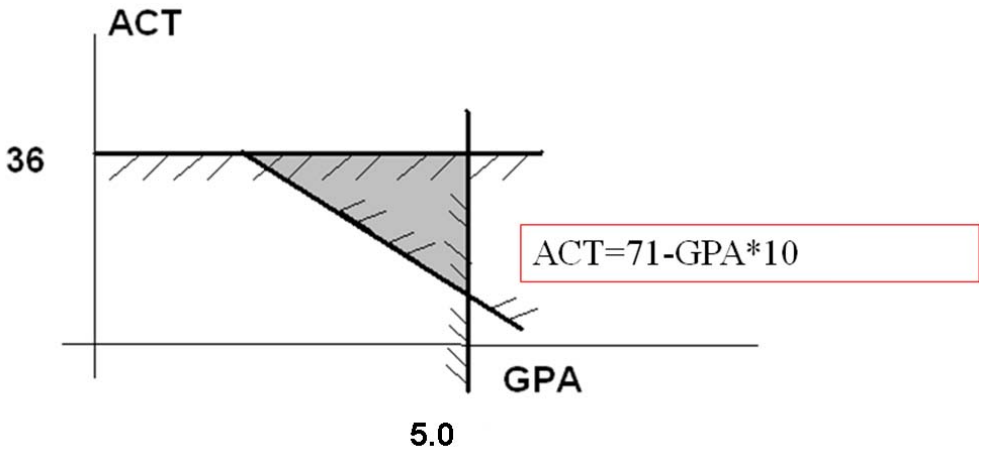
Test Case ID	Month (mm)	Day (dd)	Year (yyyy)	Expected Output
TC1	5	15	1912	Valid
TC 2	-1	20	1912	Invalid Value of Month,
TC 3	13	5	1912	Invalid Value of Month,
TC 4	6	-1	1912	Invalid Value of Day,
TC 5	8	32	1912	Invalid Value of Day,
TC 6	7	18	1810	Invalid Value of Year,
TC7	3	8	2015	Invalid Value of Year

This is a minimal solution. Of course we can extend this test suite with more classes with 30 and 31 days, with boundary value analysis, defensive testing of not numbers or some special value testing: February 28 and 29 and leap year. Extra points will be given for any of these extensions.

Q3. [12p]

Admission to Stateless University is made by considering a combination of high school grades (GPA) and ACT test scores. The entry requirements are:
ACT <= 36, GPA <=5, 10GPA+ACT >= 71.

a) Draw the valid input domain. [3p]
b) Generate test cases to test these requirements using 1x1 domain analysis process.[9p]



Variable			TC 1	TC 2	TC 3	TC 4	TC 5	TC 6
GPA	GPA <= 5.0	ON	5					
		OFF		5.1				
	Typical	In			4.7	4.8	4.8	4.6
ACT	ACT <= 36.0	ON			36			
		OFF				37		
	ACT>=71-10*GPA	ON					23	
		OFF						24
	Typical	in	34	33				
Expected result			y	n	y	n	y	n

For the OFF points the COOOOI rule has to be used.
GPA <=5.0 ON point GPA = 5, OFF point : closed border then outside the domain for example GPA= 5.1

The 1x1 domain testing matrix

Q4. [12p]

A company has the following policy for handling orders:
"All orders of non-Star clients with bad credit should be rejected.
If there is enough product in stock then the order should be accepted, otherwise order will be put in waiting list."

Design test cases using a decision table to test this policy.

Solution:

We can identify the following **conditions**:
Good credit (Yes/No)
Star client (Yes/No)
Stock sufficient (Yes/No)
an the following **actions:**

Accept, reject, waiting list
The decision table:

Conditions	Good credit	Yes	Yes	Yes	Yes	No	No	No	No
	Star client	Yes	Yes	No	No	Yes	Yes	No	No
	Stock sufficient	Yes	No	Yes	No	Yes	No	Yes	No
Actions									
	Accept	x		x		x			
	Wait		x		x		x		
	Reject							x	x

Each column is a test case.
The table can be simplified by reducing the redundant rules

Conditions	Good credit	Yes	Yes	No	No	No
	Star client	-	-	Yes	Yes	No
	Stock sufficient	Yes	No	Yes	No	-
Actions						
	Accept	x		x		
	Wait		x		x	
	Reject					x

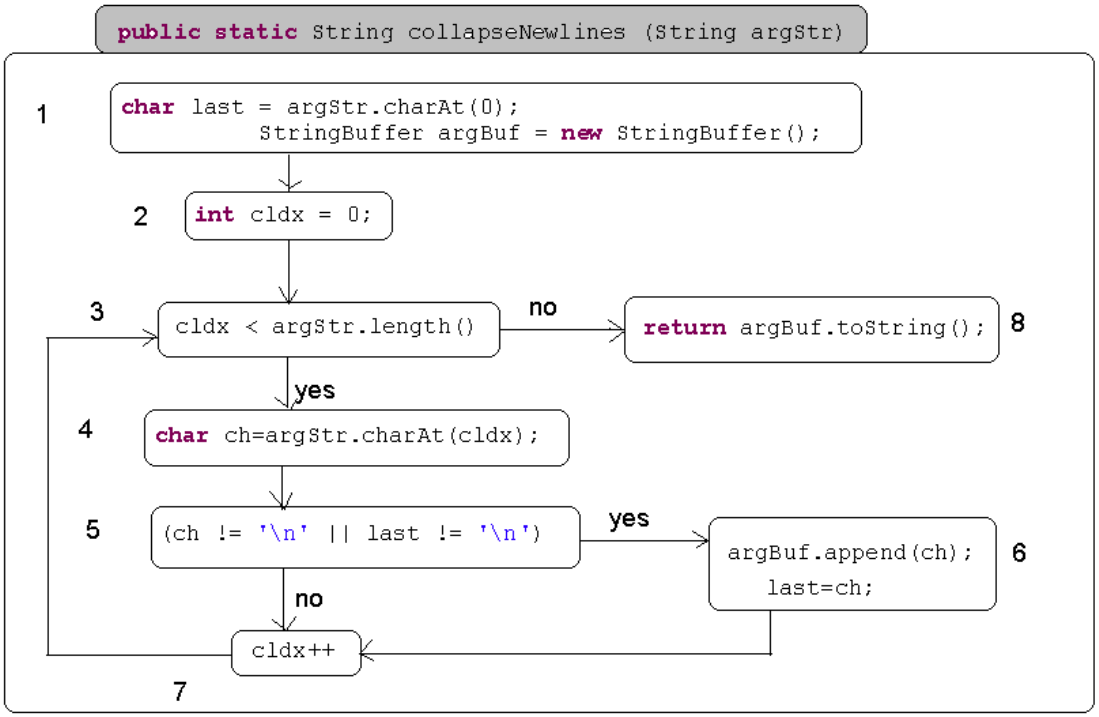
Q5. Code based testing [24p]

Given the following Java method to collapse adjacent newline characters, taken from the Apache project:

```
public static String collapseNewlines (String argStr)
         {
          char last = argStr.charAt(0);
          StringBuffer argBuf = new StringBuffer();

          for (int cldx = 0; cldx < argStr.length();cldx++)
          {
               char ch=argStr.charAt(cldx);
               if(ch != '\n' || last != '\n')
               {
                     argBuf.append(ch);
                     last=ch;
               }
          }
          return argBuf.toString();
        }
```

a) Draw the control flow graph [3p]
b) Generate a test suite that achieves 100% statement coverage. [5p]
c) Generate a test suite that is adequate with respect to the all-uses criterion [8p]
d) Generate a mutant and show a test case that will kill it. [5p]
e) Generate a mutant that will never be killed [3p]



a) the control flow graph
b) For example the test case :
input : “ab” output : “ab” will cover all the statements
c)First we summarize in a table all variables and the nodes where they are defined and used

variable	def in node	c-used in node	p=used in node	
argstr	0	1, 4	3 (3-4,3-8)	
last	1 6		5 (5-6,5-7) 5 (5-6,5-7)	
argBuf	1 6	6,8 6,8		
cldx	2 7	7,4	3(3-4, 3-8) 3(3-4, 3-8)	
ch	4	6	5(5-6,5-7)	

The paths that cover all the c-use and p-uses are:
0-1-2-3-8
0-1-2-3-4-5-7-3-8
0-1-2-3-4-5-6-7-3-8
0-1-2-3-4-5-6-7-3-4-5-7-3-8

ID	input : argstr	expected output	path
TC1	“”	“”	1-2-3-8
TC2	“\n”	“”	1-2-3-4-5-7-3-8
TC3	“a”	“a”	1-2-3-4-5-6-7-3-8
TC4	“\n\na”	“a”	1-2-3-4-5-6-7-3-4-5-7-3-8

c) A mutant is for example:

```
if(ch = '\n' || last != '\n')
```

TC4 will kill it because the output will be “\na”

d) an equivalent mutant that will never be killed is :

```
for (int cldx = 0; cldx != argStr.length();cldx++)
```