Report No	TENTA_TDDD04
Organization	Linköping University, Sweden
Last Modified	6 June 2012
Modified by	Peter Bunus (peter.bunus@liu.se)

Tentamen TDDD04 (Programvarutestning) Examination TDDD04 (Software Testing)

Tentamen: TDDD04 – Programvarutestning (2012-05-28)

Examinator: Peter Bunus

1 Information

Poängavdrag kommer att göras om punkterna nedan inte åtföljs!

1) Använd endast framsidan (delfrågor kan vara på samma sida).
2) Sortera inlämnade svar med avseende på uppgiftsnummer i stigande ordning.
3) Svaren får vara på svenska eller engelska.
4) Dina svar skall tydligt visa lösningsmetod. Enbart rätt svar kommer inte att ge poäng. I det fall du är osäker på frågeställning, skriv ner din tolkning och lös uppgiften utifrån din tolkning.

2 Betygsgränser

[0..55)	poäng	Betyg U
[55..70)	poäng	Betyg 3
[70..85)	poäng	Betyg 4
[85..100]	poäng	Betyg 5

Lycka till!

Examination: TDDD04 – Software Testing (2011-05-28)

Examiner: Peter Bunus

3 Information

Please also observe the following; otherwise it might lead to subtraction of points:

1) Use only the front side of the sheets.
2) Sort the solution sheets according to the task number.
3) Answers may be in English or Swedish.
4) Your answers should clearly show solution methods, reasons, and arguments. Short answers should be briefly motivated. If you have to a make an assumption about a question, write down the assumptions you make.

4 Grading

To pass the exam you have to do at least 55 points from 100 possible.

[0..55)	points	Grade Fx
[55..70)	points	Grade C
[70..85)	points	Grade B
[85..100]	points	Grade A

Good Luck!

Bonne chance!

Viel Glück!

Sėkmės!

祝你好運

祝福

1. Which of the following statements is NOT correct?

a) A minimal test set that achieves 100% path coverage will also achieve 100% statement coverage.
b) A minimal test set that achieves 100% path coverage will generally detect more faults than one that achieves 100% statement coverage.
c) A minimal test set that achieves 100% statement coverage will generally detect more faults than one that achieves 100% branch coverage.

Please motivate your answer by giving a short explanatory example. (10p)

Answer: C. Obviously the last answer is NOT true because statement coverage is the weakest coverage testing method compared to the branch and path coverage. The following illustrates this:

```
input x;
y = 0;
if (x>10) then
   y = 15;
end if;
z = 150/y;
```

For example, a test that has an input x = 12 will achieve statement coverage (all the statements in my program will be executed) and the test will successfully pass giving us the false impression that there are no faults in our code. However a second test with x = 5 could actually reveal that the last statement it is actually a division by zero.

---

2. Which of the following requirements is testable? (5p)

a) The system shall be user friendly.
b) The safety-critical parts of the system shall contain 0 faults.
c) The response time shall be less than one second for the specified design load.
d) The system shall be built to be portable.

Please motivate your answer. (5p)

Answer: C. This is an easy question. Since we are all engineers we know that if something is not quantified then we cannot measure it. Only variant C fulfill this criteria and we could definitely test the response time to see if it is less than 1 sec give the design load.

---

3. In prioritising what to test, the most important objective is to:

a) find as many faults as possible.
b) test high risk areas.
c) obtain good test coverage.
d) test whatever is easiest to test.

Please shortly motivate your answer. (5p)

Answer: B. We know that exhausting test is not possible therefore more “clever” testing methods need to be designed to detect faults. Several empirical studies show that “bugs” have the tendency to concentrate in a very small part of our system; therefore it is very important to

test those high-risk areas first. A good test coverage sometimes does not guarantee that high risk areas are tested.

---

4. In a system designed to work out the tax to be paid: An employee has £4000 of salary tax free. The next £1500 is taxed at 10%. The next £28000 is taxed at 22%. Any further amount is taxed at 40%. Which of these groups of numbers would fall into the same equivalence class?
a) £4800; £14000; £28000
b) £5200; £5500; £28000
c) £28001; £32000; £35000
d) £5800; £28000; £32000

Please shortly motivate your answer. (5p)

Answer: D. The figure below illustrates the input value equivalence sets together with the corresponding taxation levels.
----------4000----------5500----------33500----------
…..0%.....10%.....22%.....40%

---

5. For the example from the previous question, If, I would like to apply a boundary value testing approach which input values shall I choose for the tests.
Please shortly motivate your answer. (5p)

Answer: One should create test cased for each boundary value by choosing one point on the boundary, one point just below the boundary, and one point just above the boundary. In the previous question, in order to identify the equivalence classes you have hopefully currently identified the boundaries to be 4000, 5500, 33500 therefore the minimal boundary value testing approach will include the following input values: 3999, 4000, 4001, 5499, 5500, 5501, 33499, 33500, 335001

---

6. Which of the following is true about White and Black Box Testing Technique:-
a) Equivalence partitioning, Decision Table and Control flow are White box Testing Techniques.
b) Equivalence partitioning, Boundary Value Analysis, Data Flow are Black Box Testing Techniques.
c) Equivalence partitioning, State Transition, Use Case Testing are black box Testing Techniques.
d) Equivalence partitioning, State Transition , Use Case Testing and Decision Table are White Box Testing Techniques.

Answer: C. The correct answer is C because equivalence partitioning, state transition and use case testing are techniques that do not require any knowledge of the internal structure of the program.

---

7. Which are the minimum tests required for statement coverage and branch coverage for the following small example:

```
Read P
Read Q
If p+q > 100
then Print "Large" End if
```

```
If p > 50 then
Print "pLarge"
End if
```

a) Statement coverage is 2, Branch Coverage is 2
b) Statement coverage is 3 and branch coverage is 2
c) Statement coverage is 1 and branch coverage is 2
d) Statement Coverage is 4 and Branch coverage is 2

Please answer the question by providing the test cases for statement and branch coverage.

Answer: C
Foe achieving statement coverage one test will suffice: A test with the input values p=200 q=1 will execute all the statements.
A branch is the outcome of a decision, so branch coverage will measure which decision outcomes have been tested. There are two Boolean decisions (p+q<10 and p>50) with two possible outcomes each (true and false) therefore we will have 4 branches that would need to be covered. This could be covered by using only two tests:
Test 1: p=200 q=1
Test 2 p = 40 q=10;

---

8. The following program is used in a hypothetical retail situation. The owner of a shop has decided that her staff can have a 10 percent discount on all their purchases. If they spend more than €15, then the total discount is increased by 50 cents. The price of each item being purchased is input into the program. When -1 is entered, the total price is displayed, as well as the calculated discount and the final price to pay. For example, the values €5.50, €2.00 and €2.50 are input, equaling €10.00. The total discount would equal €1.00 (10% of €10.00), with the total price to pay equaling €9.00. A second example would have purchases of €10.50 and €5.00, equaling €15.50. In this case, as the total value is over €15, the discount would be €2.05 (10% of €15.50 is €1.55, plus 50cents as the original total is over €15), meaning that the total price to pay would be €13.45.

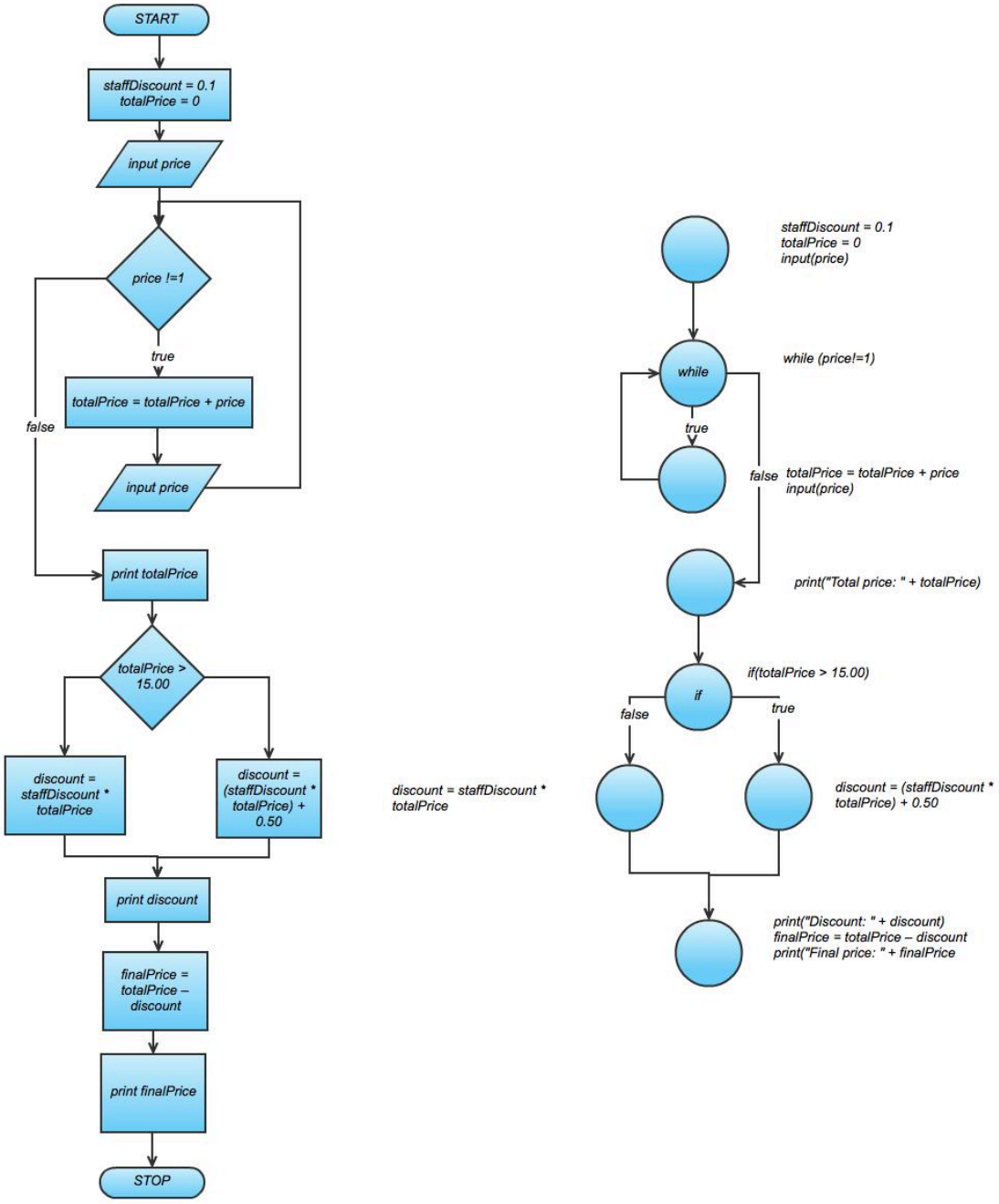
The source code, written in pseudo code, for a program which has been written to perform the task described above, is shown below:

```
1     program Example()
2     var staffDiscount, totalPrice, finalPrice, discount, price
3     staffDiscount = 0.1
4     totalPrice = 0
5     input(price)
6     while(price != -1) do
7           totalPrice = totalPrice + price
8           input(price)
9     od
10    print("Total price: " + totalPrice)
11    if(totalPrice > 15.00) then
12          discount = (staffDiscount * totalPrice) + 0.50
13    else
14          discount = staffDiscount * totalPrice
15    fi
16    print("Discount: " + discount)
17    finalPrice = totalPrice - discount
18    print("Final price: " + finalPrice)
19    endprogram
```

a) Write down the minimal set of tests that would help you to achieve statement coverage (5p)
Two tests will achieve full statement coverage:
Test 1: price = 10, price = -1 will cover lines 1,2,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19
Test 2 price = 18, price=-1 will cover lines 1,2,3,4,5,6,7,8,9,10,11,12,16,17,18,19
It is important to note that one test would require a series of inputs on the price because we would eventually like to exit the while loop.

b) Write down the minimal set of tests that would help you to achieve boundary value testing (5p)
Test 1: price = 14.99, price = -1
Test 2: price = 15, price = -1
Test 3: price = 15.01, price = -1

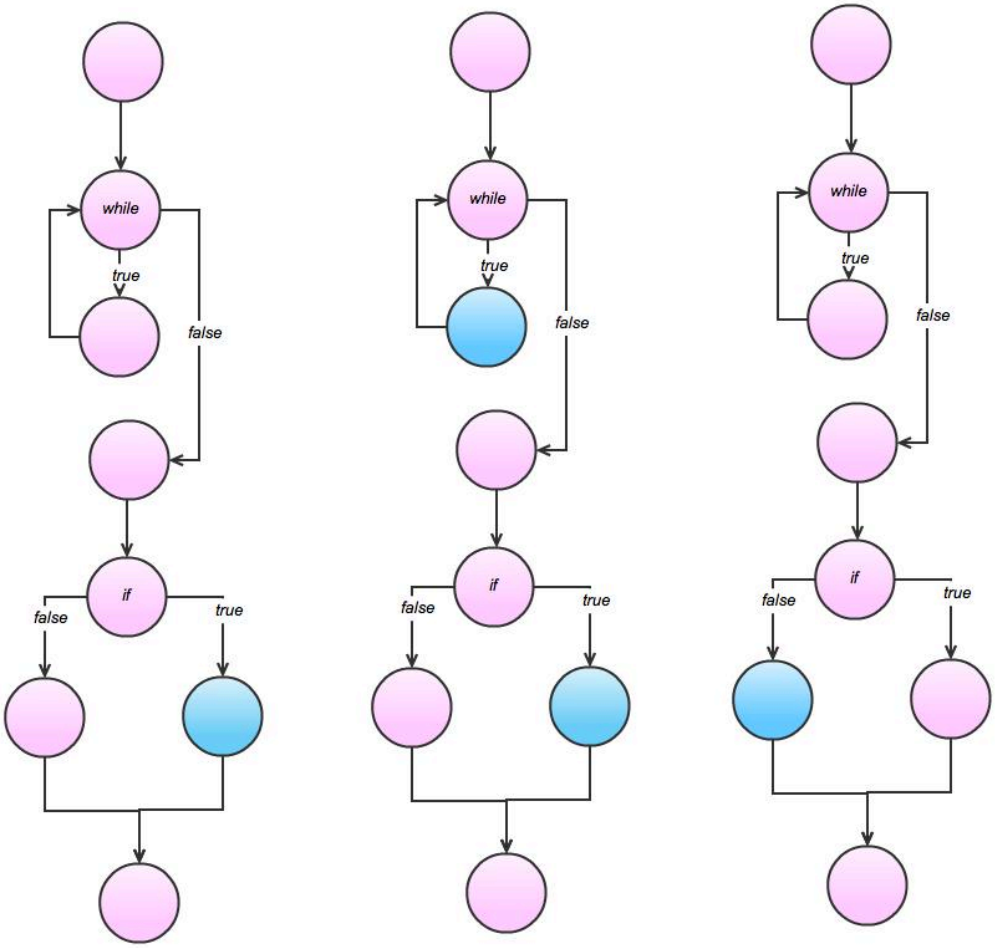
c) Draw a flow chart and a control flow graph to represent the following code: (5p+5p)
The obvious question here is what is the difference between flow charts and control flow diagrams? Flow charts are diagrams used for representing algorithms or processes.
A control flow graph is a representation using graph notation, of all paths that might be traversed through a program during its execution. In a control flow graph each node represent a basic block (a piece of code without any jumps) and the directed edges are representing jumps.



d) Calculate the cyclomatic complexity of the control graph (5p)
C= E-N+2P = 9 – 8 + 2*1 = 3

e) Write down input values for test cases that satisfy McCabe's base path coverage (5p)

The number of test cases will equal the cyclomatic complexity of the program



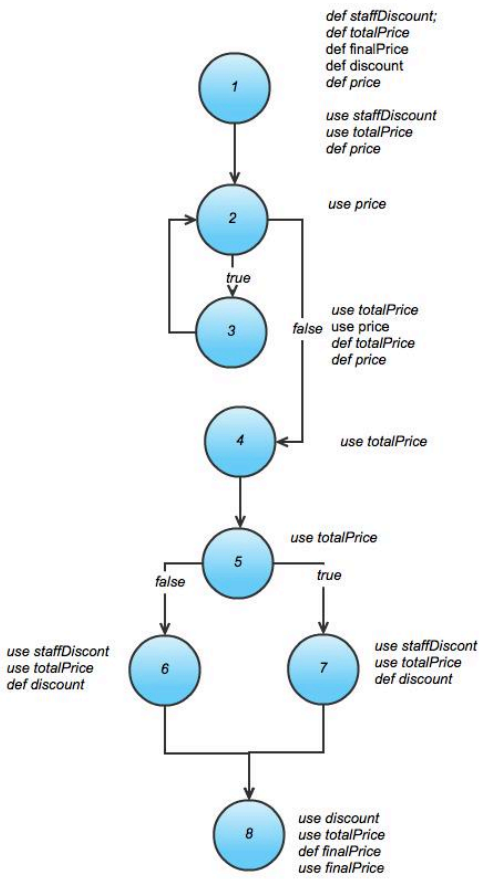
We would need to write tests that will traverse the following paths with the nodes marked with red.

Test 1:
input price = €5.50, price = €2.00, price = €2.50, price = -1
Expected output: finalPrice = €9.00

Test 2:
input price = -1
Expected output: finalPrice = €0.00
Obs: this test actually will detect a very subtle bug in the program that the discount was declared but never initialized.

Test 3:
input price = €10.50, price = €5.00, price = -1
Expected output: total price to pay = €13.45

f) Draw the data flow graph of the program and annotate the data flow graph with "definition" and "use" information of each module variable. (5p)



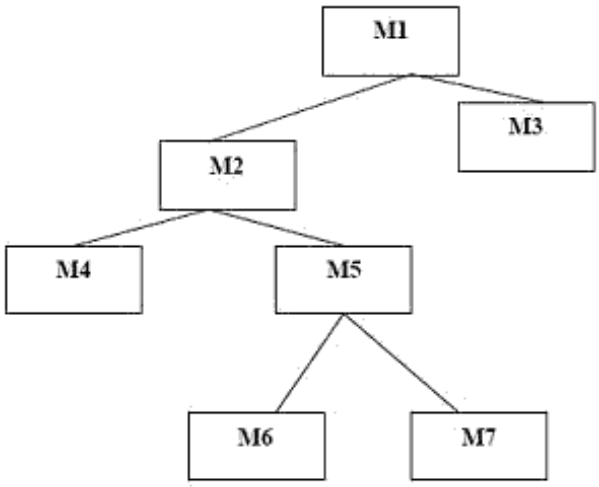
g) List all the definition-use paths (du-path) for the variable "price" and write a test for each of these test paths. (5p)

Looking at the price variable there are two defining nodes and two usage nodes.
DEF(price,1)
DEF(price,3)
USE(price,2)
USE(price,3)

Therefore there are 4 du paths
<1,2>
<1,2,3>
<3,2>
<3,2,3>

The following test could be deployed in this situation
<1,2> T1 price = -1
<1,2,3> T2: price = 10, price -1
<3,2> T2 will test this case
<3,2,3> T3: price = 10, price = 5, price = -1

13. The following figure illustrates the component hierarchy in a software system.



(a)

a. Describe the sequence of tests for integration of the components using a bottom-up approach(5p) and a top-down approach. (5p)

Bottom Up:
S1: M6, driver(M5)
S2: M7, driver(M5)
S3: M6,M7, driver(M5)
S4: M4, driver(M2)
S5: M6,M7,M5, driver(M2)
S6: M4,M6,M7,M5, driver(M2)
S7: M4,M6,M7,M5 ,M2, driver(M1)
S8: M3, driver (M1)
S8: M6,M7,M5,M4,M2, M3, M1

Top Down:
S1: M1, stub(M2), stub(M3)
S2: M1, M2, stub(M3)
S3: M1, stub(M2), M3
S4: M1, M2, stub(M4), stub(M5), M3
S5: M1, M2, M4, stub(M5), M3
S6: M1, M3, M4, M5, stub(M6), stub(M7), M3
S7: M1, M3, M4, M5, M6, stub(M7), M3
S8: M1, M3, M4, M5, stub(M6), M7, M3
S9: M1, M3, M4, M5, M6, M7, M3

b. How many stubs are needed for top-down integration? Don't forget to explain how you calculated the result, since there are different conventions of how to calculate this.(5p)

No of stubs = nodes – 1 = 7-1=6
No of sessions = (Node_leaves) + edges = 3 + 6 = 9;

c. How many drivers are needed for bottom-up integration? Motivate clearly.(5p)

No of drivers = nodes _leaves = 3
No of sessions = (Node_leaves) + edges = 3 + 6 = 9;