# Testing and Verification (DIT085)
# Solutions to Model Examination - March 2015

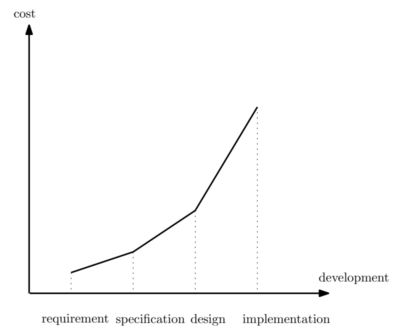
**Important Notes.** It is not allowed to use study material, computers, and calculators during the examination. The examination comprises 5 question in 2 pages. Please check beforehand whether your copy is properly printed. In order to obtain a VG you need to obtain 80/100, for a G you need to obtain 60/100. Give complete explanation and do not confine yourself to giving the final answer. The answers may be given in Dutch or English. **Good luck!**

**Exercise 1 (20 points)** Define the following concepts:

1. Validation and verification,
2. Boehm's curve,
3. Pairwise Testing,
4. Prime Path.

*Solution.*

1. Validation refers to checking compliance against intended usage (checking whether this is the correct product), while verification refer to checking the artifacts of different phases against each other, e.g., whether implementation is correct with respect to the design or the requirement specification (whether we have made the product correctly).
2. The following curve shows how the delayed detection of faults results in an increasing cause of fixing them.



3. In pairwise testing equivalence class testing is performed by covering all combinations of representatives values for each two pairs of variables.
4. A prime path is a single path that is not included in any other simple path.

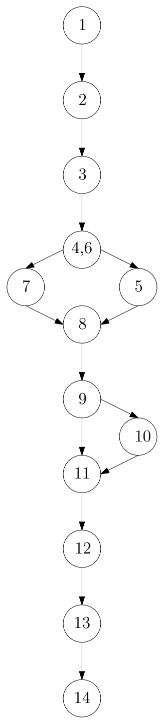
**Exercise 2 (25 points)** Consider the following program.

```
 1: Input(x);
 2: Input(y);
 3: Input(z);
 4: if y < 10 then
 5:     x := 10;
 6: else
 7:     x := y;
 8: end if
 9: if y <= z then
10:     y := y + z;
11: end if
12: y := x
13: x := y
14: write(x);
```

1. Draw the control-flow graph of the program (5 pts),
2. Calculate all definition-clear paths starting from definitions of $x$. (10 pts),
3. Define a set with the fewest number of test cases that satisfies DU-path coverage with respect to $x$. For each test case indicate the DU path that it covers. (10 pts)

*Solution.*

1. The control flow of the program is depicted below.



2. The following paths and all their proper sub-paths are definition clear paths from a definition of $x$:

   [1, 2, 3, 4, 5]

   [1, 2, 3, 6, 7]

[5, 8, 9, 11, 12]
[5, 8, 9, 10, 11, 12]
[7, 8, 9, 11, 12]
[7, 8, 9, 10, 11, 12]
[13, 14]

3. The following test cases cover the indicated DU paths:
   inputs: x=0, y = 5, z=0, output = 10, DU paths covered: [5, 8, 9, 11, 12] [13, 14]
   inputs: x=0, y = 5, z=6, output = 10, DU paths covered: [5, 8, 9, 10, 9, 11, 12] [13, 14]
   inputs: x=0, y = 15, z=0, output = 15, DU paths covered: [7, 8, 9, 11, 12] [13, 14]
   inputs: x=0, y = 15, z=20, output = 15, DU paths covered: [7, 8, 9, 10, 9, 11, 12] [13, 14]

**Exercise 3 (20 points)** Specify the following English properties in the TCTL language (input language for UPPAAL queries / properties):

1. There is no deadlock (5 pts),
2. In some execution of the model, a state can be reached in which automaton $m$ is in state $s0$ and automaton $mp$ is not in state $s1$ (5 pts),
3. In all executions, if automaton $m$ is in state $s0$ then eventually automaton $mp$ will be in state $s2$ (10 pts).

*Solution.*

1. `A[] not deadlock`,
2. `E <> (m.s0 and not mp.s1)`, and
3. `m.s0 −− > mp.s2`.

**Exercise 4 (25 points)** Consider the program given for exercise 2 Calculate $Slice(14, \{x\})$ for it. The final solution is not sufficient; you need to elaborate on the steps towards the final solution (include the relevant variables and the approximations towards the final slice). (25 pts)

*Solution.*

| **m** | **DEF(m)** | **Relevant$_0$(m)** | **Slice$_0$** | **Cond$_1$** | **Rel$_1$** | **Slice$_1$** | **Slice** |
|---|---|---|---|---|---|---|---|
| 1 | $\{x\}$ | $\emptyset$ | $\times$ | $\times$ | $\emptyset$ | $\times$ | $\times$ |
| 2 | $\{y\}$ | $\emptyset$ | $\surd$ | $\times$ | $\emptyset$ | $\surd$ | $\surd$ |
| 3 | $\{z\}$ | $\{y\}$ | $\times$ | $\times$ | $\{y\}$ | $\times$ | $\times$ |
| 4 | $\emptyset$ | $\{y\}$ | $\times$ | $\surd$ | $\{y\}$ | $\surd$ | $\surd$ |
| 5 | $\{x\}$ | $\emptyset$ | $\surd$ | $\times$ | $\emptyset$ | $\surd$ | $\surd$ |
| 6 | $\emptyset$ | $\{y\}$ | $\times$ | $\surd$ | $\{y\}$ | $\surd$ | $\surd$ |
| 7 | $\{x\}$ | $\{y\}$ | $\surd$ | $\times$ | $\{y\}$ | $\surd$ | $\surd$ |
| 8 | $\emptyset$ | $\{x\}$ | $\times$ | $\times$ | $\{x\}$ | $\times$ | $\surd$ |
| 9 | $\emptyset$ | $\{x\}$ | $\times$ | $\times$ | $\{x\}$ | $\times$ | $\times$ |
| 10 | $\{y\}$ | $\{x\}$ | $\times$ | $\times$ | $\{x\}$ | $\times$ | $\times$ |
| 11 | $\emptyset$ | $\{x\}$ | $\times$ | $\times$ | $\{x\}$ | $\times$ | $\times$ |
| 12 | $\{y\}$ | $\{x\}$ | $\surd$ | $\times$ | $\{x\}$ | $\surd$ | $\surd$ |
| 13 | $\{x\}$ | $\{y\}$ | $\surd$ | $\times$ | $\{y\}$ | $\surd$ | $\surd$ |
| 14 | $\emptyset$ | $\{x\}$ | $\times$ | $\times$ | $\{x\}$ | $\times$ | $\times$ |
| | | $\{x\}$ | | | $\{x\}$ | | |

**Exercise 5 (10 points)** Consider the following procedure, which is supposed to take an array of integers and its size and write the average of the numbers in the array (up to size) on the screen. It turns out that it outputs '8.0' when input *arr*= [2, 2, 2, 14, 8], *size*=5 is given (which is incorrect). Simplify the test-case using simplification. (Assume that you can manually check whether the outcome of each test is really correct or not.)

```
const MAX = 100;
procedure writeAvg(arr: array of integer; size: integer);
var i : integer;
    avg : real;
begin
    avg := 0;
    for i := 0 to size - 1 do
        avg := avg + arr[size - 1];
    write(avg/size);
end;
```

*Solution.*

$$ddmin([2,2,2,14,8],2) =$$
$$ddmin([14,8],2) =$$
$$14, 8$$