Technical University of Moldova

Faculty of Computers, Informatics and Microelectronics

Software Engineering and Automation Department

Software Testing

Laboratory work #4

# White Box technique

*Author:*
Vasile Drumea

*Supervisor:*
Mariana Catruc

Chișinău 2019

## 1 Purpose of the laboratory work

– Obtaining testing skills for the functionalities of a software;

– Studying the structural method of testing a program;

– Get to know White box testing technique;

## 2 Laboratory Work Requirements

– Draw the graph of the control flow of the program;

– Test the program according to the coverage criteria;

– Emphasize samples for which are obtained erroneous results for different coverage criteria;

– Comment the special cases and the tests;

– Use McCabe technique to test basic paths;

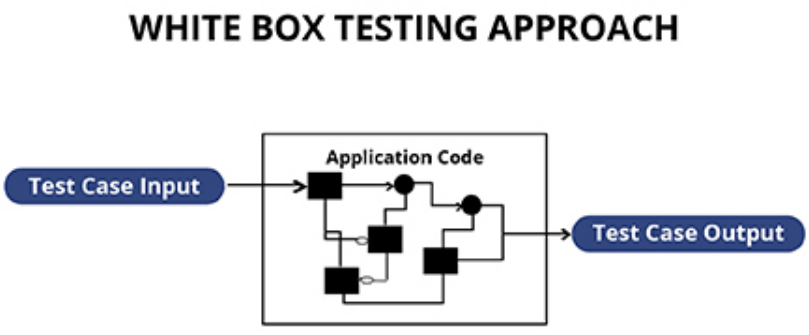
– Make conclusions and present a report;

## 3 Intro to the technique

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of software testing that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing).

In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the expected outputs.

White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today.



## 4 Laboratory work implementation

### 4.1 The chosen program

```
#include <stdio.h>
#include <string.h>
#include <algorithm>

using namespace std;

int main() {

        int T;
        scanf("%d",&T);

        while (T--) {
                int X, Sol = 0;
                scanf("%d",&X);

                while (X--) {
                        int Y;
                        scanf("%d",&Y);
                        Sol ^= Y;
                }

                if (Sol)
                        printf("DA\n");
                else
                        printf("NU\n");
        }

        return 0;
}
```

### 4.2 Task 1 - Graph of the control flow

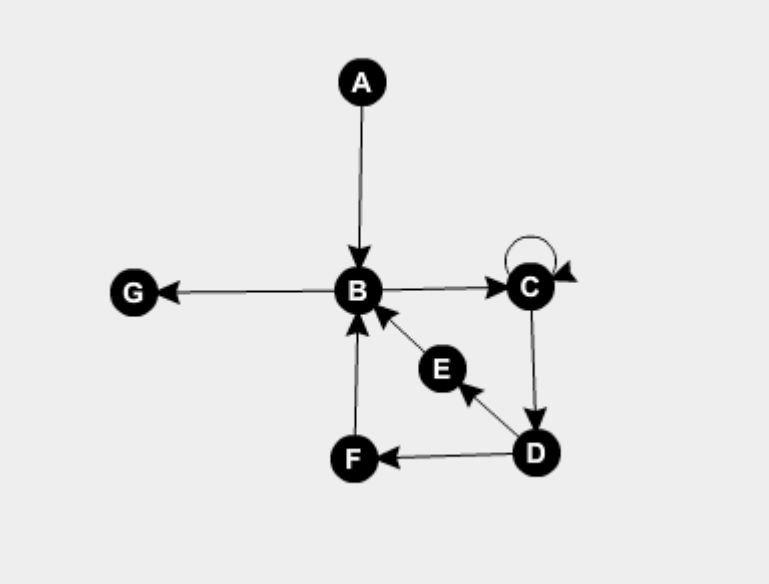



Figure 4.1– Graph of the control flow

– A - is the initial state, before we enter the **while (T–)** loop;

– B - is the start of the first while loop;

– C - comes from B and is the start of the second loop, it has self edge to represent the inner while loop;

– D - is the if statement from which we can get to E or F;

– E, F - different statements depending on if statement and from them we start a new iteration to B;

– G - is the end of the program;

### 4.3 Task 2 - Test the program according to the coverage criteria

a) All instruction coverage :

T = 2;
X1 = 1;
Y1[] = 10;
X2 = 0;
Y2[] = ;

For this test case all the statements are traversed at least once, hence each line is tested, as far as we know!!

b) All edges coverage :

T = 2;
X1 = 2;
Y1[] = 10, 2;
X2 = 0;
Y2[] = ;

To traverse each edge at least once I've added a number for the first array so that the self loop could be traversed, now all edges have been checked;

c) All simple conditions coverage :

For the conditional part in my program I have only one if statement where I check if **Sol** variable is 0 or not. In the previous cases I've covered both of them;

d) All paths coverage :

T = 2;
X1 = 2;
Y1[] = 10, 2;
X2 = 2;
Y2[] = 10, 10;

To cover all the possible paths I've added an array with length longer than 1 to have the first while loop to perform more than 1 iteration and the program works as expected;

### 4.4 Task 3 - Samples with erroneous results

I didn't met any samples to produce erroneous results. The only samples that could do so are those in which T or X are negative.

This way my program falls into infinite loop because I didn't check the values to be positive, but it is contrary to the constraints of the problem solved by the program.

### 4.5 Task 4 - Comments

– As I said previously my program falls for negative values, and thats because the iterators are only decremented and tested to be not zero but not positive;

– There are not some complex conditional samples because I have only one if statement with one condition which leads to 2 possibilities;

– Out of the four coverage methods I consider the third i.e. for the simple conditions one with a good performance and good coverage, yet it has exceptions as proven when we have more conditions;

### 4.6 Task 5 - McCabe technique

– $Nr_C = E - V + 2 = 9 - 7 + 2 = 4$;

– The paths :

  – A → B → C → D → E → B → G;

  – A → B → C → D → F → B → G;

  – A → B → C → C → D → E → B → G;

  – A → B → G;

– The test cases :

  – T = 1;
    X1 = 1;
    Y1[] = 10;

  – T = 1;
    X1 = 1;
    Y1[] = 0;

  – T = 1;
    X1 = 2;
    Y1[] = 0, 0;

  – T = 0;

– After checking the tests the program worked as expected;

## 5 Conclusion

– White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing.

– This method is named so because it is the internal part what is tested and its work flow, also the tester has access to it;

– White box testing is very thorough as the entire code and structures are tested.

– It results in the optimization of code removing error and helps in removing extra lines of code.

– Yet it has its disadvantages e.g. it is expensive, also testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.