Increasing Level of Correctness in Correlation with McCabe Complexity

N. I. Enescu, D. Mancas, E. I. Manole, and S. Udristoiu

***Abstract*—** The scope of our research is finding a correlation between the correctness indicator and the McCabe complexity indicator for software programs. For this, the correctness and McCabe complexity indicators will be calculated for a simple program, written in C programming language. The computations will be made for each program version obtained by correcting different error type found in the testing process. Will be observed there is a closed correlation between correctness and McCabe complexity in the way that for an increasing of the correctness level there will also be a significant increase of the complexity level.

***Keywords* —** correctness, complexity, cyclomatic, McCabe, correlation.

I. Informatic Solution

Software quality is defined as all the properties of a software application: technical, economical and social.

In different applications quality characteristics also have different roles, depending on the application purpose.

Complexity is the quality characteristic for which were developed most of the metrics systems and is studied in correlation with other characteristics; complexity determines also the increase of the price for trading informatics applications, as well as the efforts to develop new versions or to rewrite the applications or only some components.

In case of informatics applications complexity is an important factor. The number of defects is proportional with the complexity of the programs systems. This determines an increase of the difficulties in the programming tasks.

Correctness is the quality characteristic that is the hardest to obtain. For software products with a high complexity, a complete testing is impossible to be made, in order to offer the assurance that all error have been found and corrected.

Application correctness is parted in four categories:

- *Syntactical correctness* which presumes that the program is correct when it compiles without errors; in other words, during runtime; when a program is syntactically correct, it is restricted to the code and language in use
- *Functional correctness* presumes that a program is correct when it satisfies the specifications
- *Design correctness*, presumes that the program is correctly structured so that it can permit extensions; in this case, the experience in designing applications is the key to a correct program structure;
- *Performance correctness and the validation and verification of inputs and outputs:* it presumes that the program must send outputs adequate with the valid inputs and it also has to be optimized regarding the cod length and the running speed.

II. Defining Used Formulas

Correctness of an application is marked out in testing [6].

It means that the data test SDT1, SDT2, ... SDTNT need to obtain results RT1, RT2, ... RTNT. In reality, in the process of testing to identify situations in which results program SDTi set of data which is different from RPi result in RTi given specifications.

For the team that develops software, a result categorically related to the accuracy or incorrectness. Application information is irrelevant in relation to post-test costs. It is therefore necessary to define an indicator of correctness ICP defined the interval [0, 1].

If ICP = 0 result that all data on test results led to different results RPi specifications RTi of the whole range of types of errors.

If ICP = 1 result that all data test only led to results identical to RTi without registering errors.

It means that for the ICP belongs interval (0, 1) shall be established an aggregation which take into account errors.

It means that for the ICP belongs interval (0, 1) shall be established an aggregation which take into account errors.

Still, there are errors on levels of aggregation ERR11, ERR12, ... ERRij, ... ERRNE $_{NT}$.

For each type of error is given an important factor of PC1, PC2, ... PCNE.

In [6] it is defined the indicator of correctness by ICP relationship:

$$ICP = \begin{cases} 0, \text{daca} \sum_{j=1}^{NT} \sum_{h=1}^{NE} ERR_{hj} = 0 \\ \\ \dfrac{\sum_{j=1}^{NT} \sum_{h=1}^{NE} PC_h \cdot ERR_{hj}}{\sum_{j=1}^{NT} \sum_{h=1}^{NE} ERR_{hj}}, \text{else} \end{cases} \quad (1)$$

Complexity in McCabe way or Cyclomatic Complexity of a program is given in [4] by:

CM = na – nn + 2 (2)

where:

- na – number of arcs
- nn – nodes number in the graph associated to the program

The obtained result is compared with a set of threshold values, given in table 1:

Cyclomatic complexity	**Risk evaluation**
1- 10	Simple program/module, low risk
11- 20	Complex program/module, middle risk
21 – 50	Very complex program/ module, high risk
Bigger then 50	Untestable program/module, very high risk

Table 1. Threshold values for complexity by [5]Please submit your manuscript electronically for review as e-mail attachments.

III. Realized experiment

There is considered a program, which, related to n and m variables, which are in [0, 10] interval, it computes:

a) $\dfrac{1}{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}}$ where a_{ij} are the integer coefficients of the A matrix, with n lines and m columns, and $a_{ij}\in$[-100, 100], if $n \neq m$

b) $\dfrac{\sum_{i=1}^{n} a_{ii}}{\sum_{i=1}^{n} a_{i,n-i}}$ where a_{ij} are the integer coefficients of the square A matrix and $a_{ij}\in$[-10, 10], if n = m

c) $\dfrac{1}{\prod_{i=0}^{k} v_i}$ where v_i are integer elements of the V vector and $v_i\in$[-10, 10] and k=n if m=0 or k=m if n=0, if n = 0 or m = 0

d) $\sqrt{b/c}$ where b and c are integer numbers, if n = m = 1

The result has to be displayed as a number with a maximum of 2 decimals.

The program has 4 functions: M1, M2, M3 and M4 corresponding to the four points of the problem.

There are considered four test sets for the PROG program.

The test set 1, SDT1, table 2.

Test	Input data	Expected result
T_{11}	n = -1; m = 3	Error: n outside the range
T_{12}	n = 2; m = 11	Error: m outside the range
T_{13}	n = -2; m = 15	Error: n and m outside the range
T_{14}	n = 3; m = 2 $A=\begin{pmatrix}1 & 10\\ 0 & -101\\ 1 & 2\end{pmatrix}$	Error: -101 outside the range
T_{15}	n = 2; m = 2 $A=\begin{pmatrix}102 & 2\\ 1 & 10\end{pmatrix}$	Error: 102 outside the range
T_{16}	n = 3; m = 2 $A=\begin{pmatrix}0 & 0\\ 0 & 0\\ 0 & 0\end{pmatrix}$	Error: elements sum is zero
T_{17}	n = 3; m = 2 $A=\begin{pmatrix}1 & 1\\ 0 & 1\\ 1 & 1\end{pmatrix}$	Result is 0.20

Table 2. Test set for M1 function

Test set 3, SDT_2, table 3.

Test	Input data	Expected result
T_{21}	n = m = -1	Error: n and m outside the range
T_{22}	n = m = 3 $A=\begin{pmatrix}10 & 4 & 5\\ 6 & -9 & -101\\ 101 & 8 & 7\end{pmatrix}$	Error: -101, 101 outside the range
T_{23}	n = m = 3 $A=\begin{pmatrix}1 & 8 & -9\\ 1 & -1 & 5\\ 10 & 9 & 5\end{pmatrix}$	Error: the sum of the elements from the secondary diagonal is 0
T_{24}	n = m = 3	The result is 0.00

	$A = \begin{pmatrix} -1 & 8 & 10 \\ 9 & 1 & 5 \\ 0 & 8 & 0 \end{pmatrix}$	
T_{25}	n = m = 3 $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \end{pmatrix}$	The result is 0.80

Table 3. Test set for M2 function

Test set 4, SDT_3, table 4.

Test	Input data	Expected result
T_{31}	n = 0; m = -1	Error: m outside the range
T_{32}	n = 11; m = 0	Error: n outside the range
T_{33}	n = 0; m = 3 $V = \begin{pmatrix} 11 & 1 & -1 \end{pmatrix}$	Error: 11 outside the range
T_{34}	n = 0; m = 3 $V = \begin{pmatrix} 0 & 5 & -1 \end{pmatrix}$	Error: elements product is 0
T_{35}	n = 3; m = 0 $V = \begin{pmatrix} 1 & -3 & 2 \end{pmatrix}$	The result is -0.16

Table 4. Test set for M3 function

Test set 5, SDT_4, table 5.

Test	Input data	Expected result
T_{41}	n = m = 1; b = 5; c = 0	Error: c is 0
T_{42}	n = m = 1; b = -1; c = 3	Error: -b/c is less than 0
T_{43}	n = m = 1; b = 1; c = -1	Error: -b/c is less than 0
T_{44}	n = m = 1; b = 0; c = 5	Result is 0.00
T_{45}	n = m = 1; b = 5; c = 5	Result is 1.00

Table 5. Test set for M4 function

The program PROG1 is realized without checking the correctness.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
float M1(int n, int m){
        int a[10][10], i, j, s;
        printf("The elements for A:");
        s = 0;
        for(i = 0; i < n; i++){
                for(j = 0; j < m; j++){
  printf("a[%d][%d]=", i, j);
                        scanf("%d",&a[i][j]);
                        s += a[i][j];
                }
        }
        return 1/s;
}
float M2(int n){
        int a[10][10], i, j, s1, s2;
        printf("The elements for A:");
        s1 = 0;
        s2 = 0;
        for(i = 0; i < n; i++){
                for(j = 0; j < n; j++){
                        printf("a[%d][%d]=", i, j);
                        scanf("%d",&a[i][j]);
                }
        }
        for(i = 0; i < n; i++){
                s1 += a[i][i];
                s2 += a[i][n-i-1];
        }
return s1/s2;
}
float M3(int k){
        int v[10], i, p;
        printf("The elements for V:");
        p = 1;
        for(i = 0; i < k; i++){
                printf("v[%d]=", i);
                scanf("%d",&v[i]);
                p *= v[i];
        }
return 1/p;
}
float M4(){
        int b, c;
        printf("b=");
        scanf("%d", &b);
        printf("c=");
        scanf("%d", &c);
        return sqrt(b/c);
}
void main(){
        int n, m;
        float rez = 0;
        printf("n=");
        scanf("%d", &n);
        printf("m=");
        scanf("%d", &m);
        if (n != m){
                if (n == 0 || m == 0)
  rez = M3(n == 0? m: n);
                else
                        rez = M1(n, m);
        }
        else{
            if(n == 1)
                rez = M4();
            else
                rez = M2(n);
        }
        printf("The result is %.2f", rez);
    }
```

After executing PROG1 with the input data given in tables 2, 3,4 and 5 the table 6 is obtained.

Test	Program result	Expected result
T_{11}	Exception: Integer division by zero	Error: n outside the range
T_{12}	Result is 0.00	Error: m outside the range
T_{13}	Exception: Integer division by zero	Error: n and m outside the range
T_{14}	Result is 0.00	Error: -101 outside the range
T_{15}	Result is 37.00	Error: 102 outside the range
T_{16}	Exception: Integer division by zero	Error: elements' sum is 0
T_{17}	Result is 0.00	The result is 0.20
T_{21}	Exception: Integer division by zero	Error: n and m outside the range
T_{22}	Result is 0.00	Error: -101. 101 outside the range
T_{23}	Exception: Integer division by zero	Error: the sum of the elements from the secondary diagonal is 0
T_{24}	Result is 0.00	Result is 0.00
T_{25}	Result is 0.00	Result is 0.80
T_{31}	Result is 1.00	Error: n or m outside the range
T_{32}	Exception: Access violation reading location 0x0000000b	Error: m or n outside the range
T_{33}	Result is 0.00	Error: 11 outside the range
T_{34}	Exception: Integer division by zero	Error: elements' product is 0
T_{35}	Result is 0.00	Result is -0.16
T_{41}	Exception: Integer division by zero	Error: c is 0
T_{42}	Result is 0.00	Error: -b/c is less than 0
T_{43}	Result is -1.00	Error: -b/c is less than 0
T_{44}	Result is 0.00	Result is 0.00
T_{45}	Result is 1.00	Result is 1.00

Table 6. Running time result

It is observed that after the runtime there are two types of errors:

- Exceptions, noted with E_1, for which a 0.7 gravity coefficient is assigned
- Wrong results, noted with E_2, for which a 0.3 gravity coefficient is assigned

So, for the given test data the table 7 is obtained:

Test set	E_1	E_2
SDT_1	3	4
SDT_2	2	2
SDT_3	2	3
SDT_4	1	2
Total	8	11

Table 7. Error number found for the test sets

The correctness indicator, ICP_1, for the PROG1 program is

$$ICP_1 = \frac{0,7*8+0,3*11}{8+11} = \frac{5,6+3,3}{19} = 0,46 \tag{3}$$

The graph structure for the PROG1 is presented in figure 1:

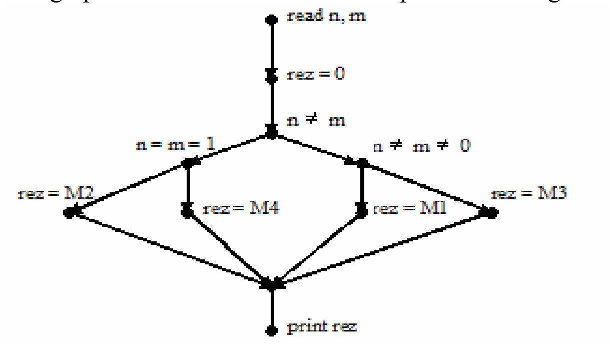

Figure 1. The graph structure for the PROG1

For M1 function rezult the graph structure from figure 2:

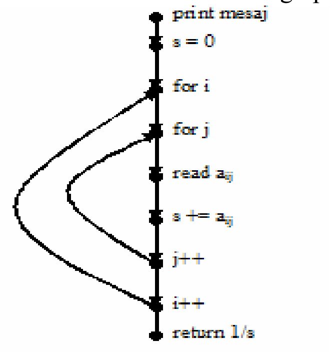

Figure 2. The graph structure for M1 module

For M2 function rezult the graph structure from figure 3:

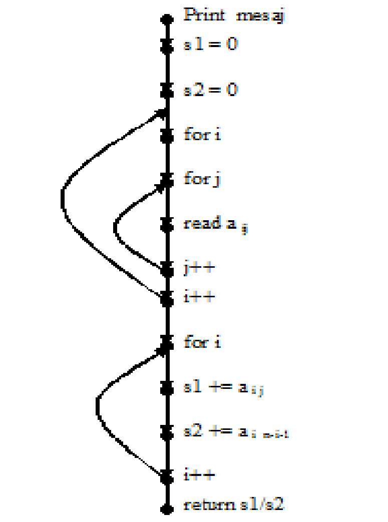

Figure 3. The graph structure for M2 module

For M3 function rezult the graph structure from figure 4:

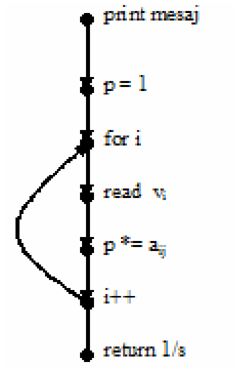

Figure 4. The graph structure for M3 module

For M4 function rezult the graph structure from figure 5:

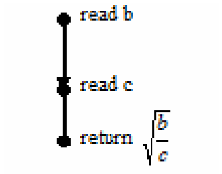

Figure 5. The graph structure for M4 module

The McCabe complexity for PROG1 is computed in table 8 and shows the differences between modules complexity.

	na	nn	CM
M1	10	9	3
M2	15	13	4
M3	7	7	2
M4	2	3	1
main	13	11	4
PROG1	47	43	6

Table 8. McCabe complexity for PROG1

Another program is realized trying to eliminate E1 type errors. The program is PROG2.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
float M1(int n, int m){
        int a[10][10], i, j, s;
        int error = 0;
        int err[100], l;
        if (n <= 0 || n > 10){
                error = 10;
        }
        if (m <= 0 || m > 10){
                error += 100;
        }
        if (error > 0){
                if (error > 100){
           printf("Error: n and m outside the
                   range");
                } else if (error == 100){
           printf("Error: m outside the
                range");
        } else {
                printf("Error: n outside the
                        range");
                }
                return -1000;
        }
        printf("Elements of A matrix:");
        s = 0;
        l = 0;
        for(i = 0; i < n; i++){
                for(j = 0; j < m; j++){
                        printf("a[%d][%d]=", i, j);
                        scanf("%d",&a[i][j]);
                  if (a[i][j] < -100 || a[i][j] > 100){
   err[l] = a[i][j];
                                      l++;
                }
                s += a[i][j];
            }
        }
        if (l > 0){
                printf("Error: ");
                for(i = 0; i < l; i++)
                        printf("%d, ", err[i]);
                printf(" outside the range");
                return -1000;
        }
        if (s == 0){
            printf("Error: elements' sum is 0");
            return -1000;
        }
        return 1/s;
}

float M2(int n){
        int a[10][10], i, j, s1, s2;
        int err[100], l;
        if (n <= 0 || n > 10){
                printf("Error: n and m outside
                        the range ");
                return -1000;
        }
        printf("Elements of A matrix:");
        s1 = 0;
        s2 = 0;
        l = 0;
        for(i = 0; i < n; i++){
                for(j = 0; j < n; j++){
                    printf("a[%d][%d]=", i, j);
                    scanf("%d",&a[i][j]);
                    if (a[i][j] < -10 || a[i][j] > 10){
    err[l] = a[i][j];
                                l++;
                }
            }
        }
        if (l > 0){
                printf("Error: ");
                for(i = 0; i < l; i++)
                        printf("%d, ", err[i]);
                printf("outside the range ");
                return -1000;
        }
        for(i = 0; i < n; i++){
                s1 += a[i][i];
                s2 += a[i][n-i-1];
        }
        if (s2 == 0){
                printf("Error: elements' sum from the
first diagonal is 0");
                return -1000;
        }
        return s1/s2;
    }

    float M3(int k){
        int v[10], i, p;
        int err[10], l;
        if (k < 0 || k > 10){
                printf("Error: n or m outside
                        the range ");
                return -1000;
        }
        printf("Elements of vector V:");
        p = 1;
        l = 0;
        for(i = 0; i < k; i++){
                printf("v[%d]=", i);
                scanf("%d",&v[i]);
                if (v[i] < -10 || v[i] > 10){
                        err[l] = v[i];
                        l++;
                }
                p *= v[i];
        }
        if (l > 0){
                printf("Error: ");
                for(i = 0; i < l; i++)
                        printf("%d, ", err[i]);
                printf("outside the range ");
                return -1000;
        }
        if (p == 0){
                printf("Error: elements'
                  product is 0");
                return -1000;
        }
        return 1/p;
}
float M4(){
        int b, c;
        printf("b=");
        scanf("%d", &b);
        printf("c=");
        scanf("%d", &c);
        if (c == 0){
                printf("Error: c is 0");
                return -1000;
        }
        if (b/c < 0){
                printf("Error: -b/c is
                    negative");
                return -1000;
        }
        return sqrt(b/c);
}

void main(){
        int n, m;
        float rez = 0;
        printf("n=");
        scanf("%d", &n);
        printf("m=");
        scanf("%d", &m);
        if (n != m){
                if (n == 0 || m == 0)
                        rez = M3(n == 0? m: n);
                else
                        rez = M1(n, m);
                }
        else{
                if(n == 1)
                        rez = M4();
                else
                        rez = M2(n);
                }
        if (rez != -1000)
                printf("The result is  %.2f", rez);
}
```

After runtime, with the input data from 2, 3,4 and 5 tables, the table 9 is obtained.

Test	Program's result	Expected result
T_{11}	Error: n outside the range	Error: n outside the range
T_{12}	Error: m outside the range	Error: m outside the range
T_{13}	Error: n and m outside the range	Error: n and m outside the range
T_{14}	Error: -101 outside the range	Error: -101 outside the range
T_{15}	Error: 102 outside the range	Error: 102 outside the range
T_{16}	Error: elements' sum is 0	Error: elements' sum is 0
T_{17}	Rersult is 0.00	Rersult is 0.20
T_{21}	Error: n and m outside the range	Error: n and m outside the range
T_{22}	Error: -101 and 101 outside the range	Error: -101 and 101 outside the range
T_{23}	Error: the sum of the elements from the secondary diagonal is 0	Error: the sum of the elements from the secondary diagonal is 0
T_{24}	Result is 0.00	Result is 0.00
T_{25}	Result is 0.00	Result is 0.80
T_{31}	Error: n or m outside the range	Error: n or m outside the range
T_{32}	Error: n or m outside the range	Error: n or m outside the range
T_{33}	Error: 11 outside the range	Error: 11 outside the range
T_{34}	Error: elements' product is 0	Error: elements' product is 0
T_{35}	Result is 0.00	Result is -0.16
T_{41}	Error: c is 0	Error: c is 0
T_{42}	Result is 0.00	Error: -b/c is less than 0
T_{43}	Result is 0.00	Error: -b/c is less than 0
T_{44}	Result is 0.00	Result is 0.00
T_{45}	Result is 1.00	Result is 1.00

Table 9. Runtime result for PROG2

It can be observed that after the execution there is only one type of error, E2, with a 0.3 coefficient.

So, for the given data sets, we have table 10.

Test set	E_1	E_2
SDT_1	0	1
SDT_2	0	1
SDT_3	0	1
SDT_4	0	3
Total	0	6

Table 10. Errors found for the PROG2 with the given data set

The correctness indicator, ICP_2, for PROG2 program is:

$$ICP_2 = \frac{0,7 * 0 + 0,3 * 6}{6} = \frac{1,8}{6} = 0,30 \quad (4)$$

The graph structure for the PROG2 is presented in figure 6:

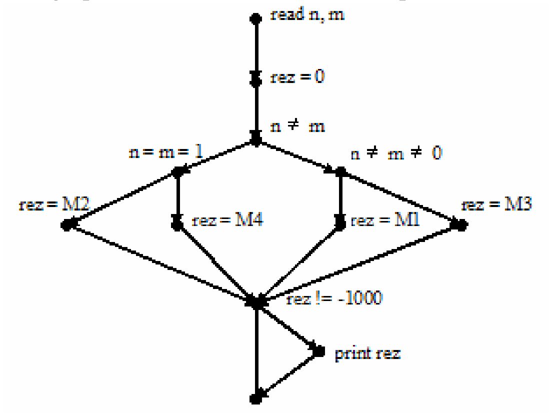

Figure 6. The graph structure for the PROG2

For M1 function rezult the graph structure from figure 7:

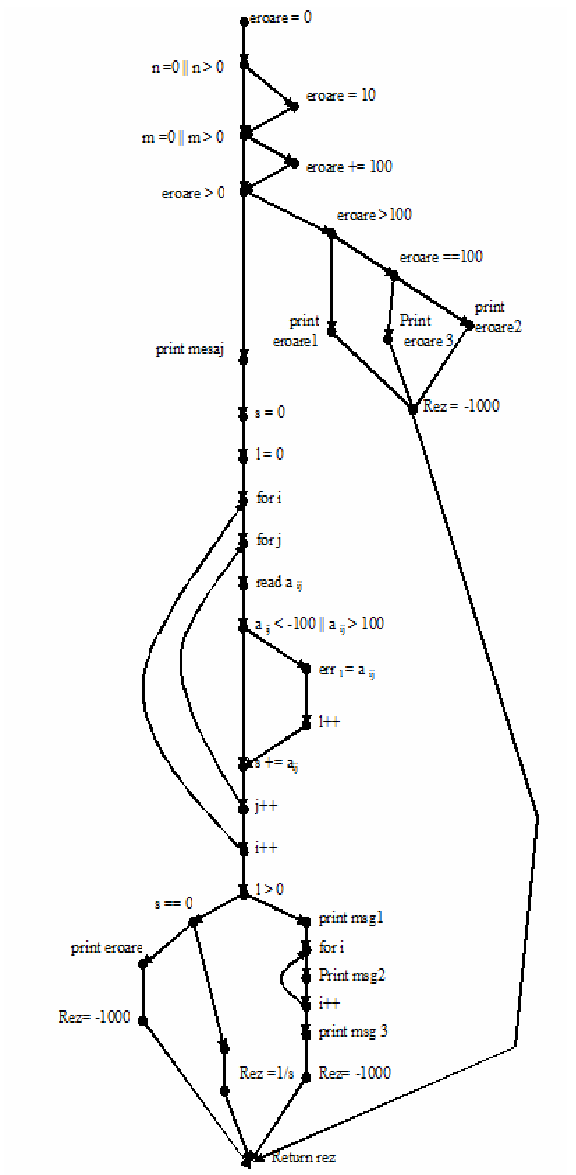

Figure 7. The graph structure for M1 module

For M2 function rezult the graph structure from figure 8:

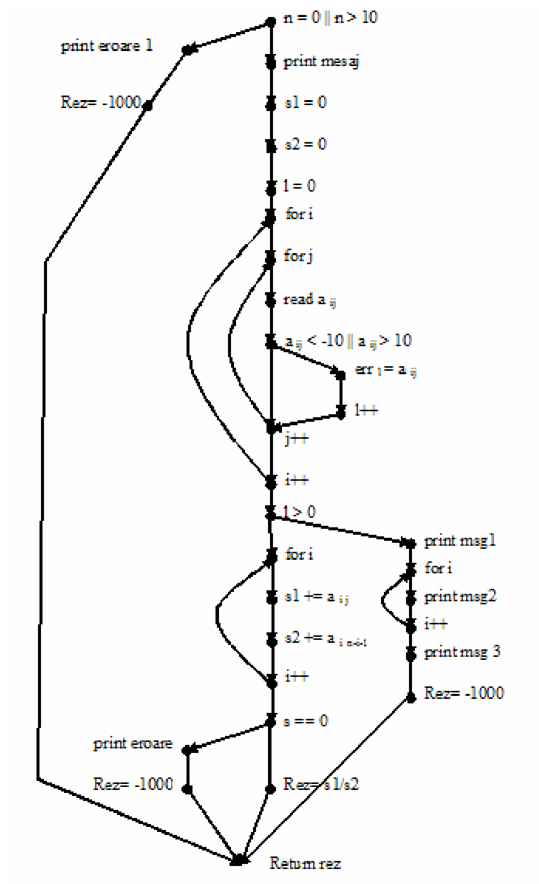

Figure 8. The graph structure for M2 module

For M3 function rezult the graph structure from figure 9:

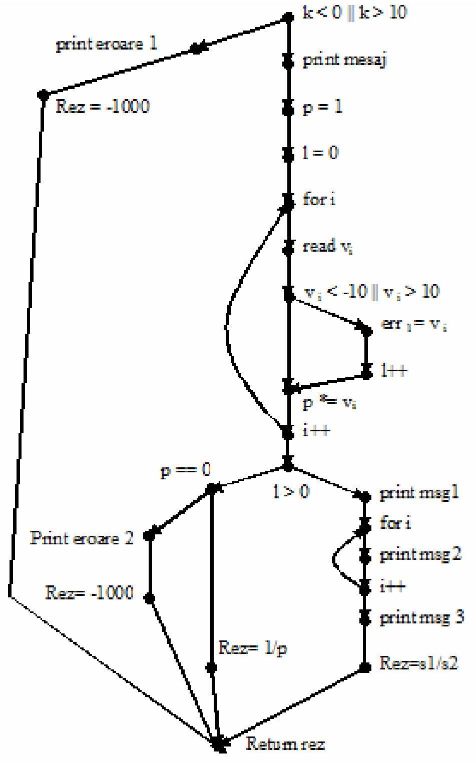

Figure 9. The graph structure for M3 module

For M4 function rezult the graph structure from figure 10:

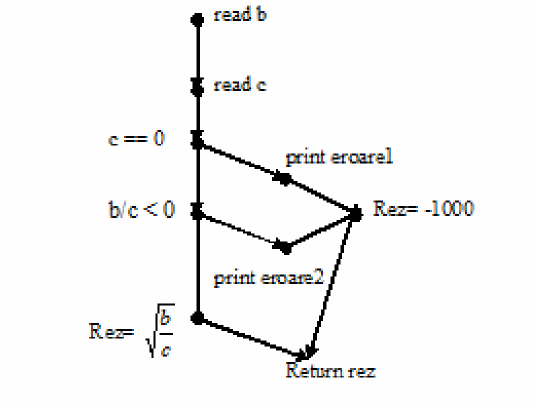

Figure 10. The graph structure for M4 module

The McCabe complexity for PROG2 is computed in table 11 and shows the differences between modules complexity.

	na	nn	CM
M1	43	36	9
M2	34	30	6
M3	26	24	4
M4	8	8	2
main	15	12	5
PROG2	126	110	18

Table 11. McCabe complexity for PROG2

Another program created by eliminating E2 type errors and is named PROG3.

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
float M1(int n, int m){
          int a[10][10], i, j, s;
          int error = 0;
          int err[100], l;
          if (n <= 0 || n > 10){
                    error = 10;
          }
          if (m <= 0 || m > 10){
                    error += 100;
          }
          if (error > 0){
                    if (error > 100){
                              printf("Error: n and m
   outside the range ");
                    } else if (error == 100){
                              printf("Error: m is outside
                                        the range ");
          } else {
                    printf("Error: n is outside the
                              range ");
          }
          return -1000;
          }
          printf("A matrix elements:");
          s = 0;
          l = 0;
          for(i = 0; i < n; i++){
                    for(j = 0; j < m; j++){
                      printf("a[%d][%d]=", i, j);
                      scanf("%d",&a[i][j]);
                      if (a[i][j] < -100 || a[i][j] > 100){
                              err[l] = a[i][j];
                              l++;
                    }
                    s += a[i][j];
                  }
          }
          if (l > 0){
                    printf("Error: ");
                    for(i = 0; i < l; i++)
                              printf("%d, ", err[i]);
                    printf("outside the range ");
                   return -1000;
          }
          if (s == 0){
          printf("Error: elements' sum is 0");
          return -1000;
    }
return 1.0/s;
}

float M2(int n){
          int a[10][10], i, j, s1, s2;
          int err[100], l;
          if (n <= 0 || n > 10){
                    printf("Error: n and m outside the
                              range ");
                    return -1000;
          }
          printf("A matrix elements:");
          s1 = 0;
          s2 = 0;
          l = 0;
          for(i = 0; i < n; i++){
                    for(j = 0; j < n; j++){
                           printf("a[%d][%d]=", i, j);
                          scanf("%d",&a[i][j]);
   if (a[i][j] < -10 || a[i][j] > 10){
                              err[l] = a[i][j];
                              l++;
                         }
                    }
          }
          if (l > 0){
                    printf("Error: ");
                    for(i = 0; i < l; i++)
                              printf("%d, ", err[i]);
```

```
                printf("outside the range ");
                return -1000;
        }
        for(i = 0; i < n; i++){
                s1 += a[i][i];
                s2 += a[i][n-i-1];
        }
        if (s2 == 0){
                printf("Error: elements' sum from
                   the secondary diagonal is 0");
                return -1000;
        }
        return (1.0 * s1)/s2;
}

float M3(int k){
        int v[10], i, p;
        int err[10], l;
        if (k < 0 || k > 10){
               printf("Error: n or m outside the
                        range ");
                return -1000;
        }
        printf("V vector's elements:");
        p = 1;
        l = 0;
        for(i = 0; i < k; i++){
                printf("v[%d]=", i);
                scanf("%d",&v[i]);
                if (v[i] < -10 || v[i] > 10){
                        err[l] = v[i];
                        l++;
                }
                p *= v[i];
        }
        if (l > 0){
                printf("Error: ");
                for(i = 0; i < l; i++)
                        printf("%d, ", err[i]);
                printf("outside the range ");
                return -1000;
        }
        if (p == 0){
                printf("Error: elements' product is
                        0");
                return -1000;
        }
        return 1.0/p;
}

float M4(){
        int b, c;
        printf("b=");
        scanf("%d", &b);
        printf("c=");
        scanf("%d", &c);
        if (c == 0){
                printf("Error: c is 0");
                return -1000;
        }
        if ((1.0 * b)/c < 0){
                printf("Error: -b/c is negative");
                return -1000;
        }
        return sqrt((1.0 * b)/c);
}

void main(){
        int n, m;
        float rez = 0;
        printf("n=");
        scanf("%d", &n);
        printf("m=");
        scanf("%d", &m);
        if (n != m){
                if (n == 0 || m == 0)
                        rez = M3(n == 0? m: n);
                else
                        rez = M1(n, m);
        }
        else{
                if(n == 1)
                        rez = M4();
                else
                        rez = M2(n);
        }
        if (rez != -1000)
    printf("The result is %.2f", rez);
}
```

After runtime, having the given test data sets defined in tables 1, 2, 3 and 4 we obtain table 12.

Test	Program's result	Expected result
T_{11}	Error: n out of range	Error: n out of range
T_{12}	Error: m out of range	Error: m out of range
T_{13}	Error: n and m out of range	Error: n and m out of range
T_{14}	Error: -101 out of range	Error: -101 out of range
T_{15}	Error: 102 out of range	Error: 102 out of range
T_{16}	Error: elements' sum is 0	Error: elements' sum is 0
T_{17}	Result is 0.20	Result is 0.20
T_{21}	Error: n and m out of range	Error: n and m out of range
T_{22}	Error: -101, 101	Error: -101, 101 out

	out of range	of range
T_{23}	Error: elements' sum from the secondary diagonal is 0	Error: elements' sum from the secondary diagonal is 0
T_{24}	Result is 0.00	Result is 0.00
T_{25}	Result is 0.80	Result is 0.80
T_{31}	Error: n or m out of range	Error: n or m out of range
T_{32}	Error: n or m out of range	Error: n or m out of range
T_{33}	Error: 11 out of range	Error: 11 out of range
T_{34}	Error: elements' product is 0	Error: elements' product is 0
T_{35}	Result is -0.16	Result is -0.16
T_{41}	Error: c is 0	Error: c is 0
T_{42}	Error: -b/c is less than 0	Error: -b/c is less than 0
T_{43}	Error: -b/c is less than 0	Error: -b/c is less than 0
T_{44}	Result is 0.00	Result is 0.00
T_{45}	Result is 1.00	Result is 1.00

Table 12. $PROG_3$ program result after runtime

It can be observed that after running the tests, there are no more errors.

The correctness indicator, ICP_3, for $PROG_3$ program is 0.

CM for $PROG_3$ is identical with CM for $PROG_2$.

So the level of correctness and McCabe complexity programs PROG1, PROG2 and PROG3 is presented in Table 13 and Figure 11.

	Index of correctness (ICP)	McCabe Complexity (CM)
$PROG_1$	0,46	6
$PROG_2$	0,30	18
$PROG_3$	0	18

Table 13. The level of accuracy and reliability programs PROG1, PROG2 and PROG3

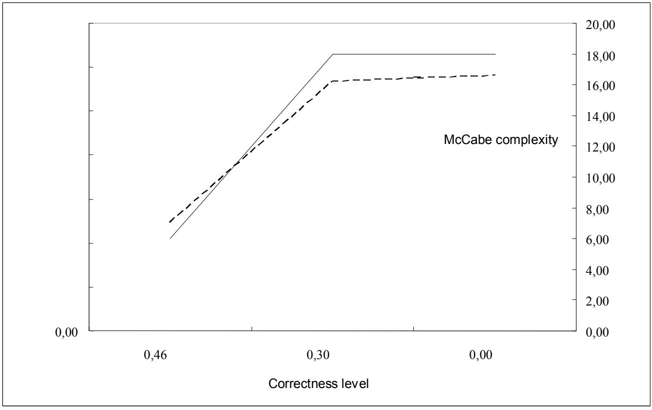

Figure 11. The relationship between correctness and McCabe complexity

The coefficient of correlation between the level of correctness and McCabe complexity CC is calculated by the formula:

$$CC_2 = \frac{\sum_{i=1}^{NV} \left(ICP_i - \overline{ICP}\right) \cdot \left(CM_i - \overline{CM}\right)}{NV \cdot S_{ICP} \cdot S_{CM}} \qquad (5)$$

where:

NV - represents the number of variations of the program

-- The index represents the average accuracy given by:

$$\overline{ICP} = \frac{\sum_{i=1}^{NV} ICP_i}{NV} \qquad (6)$$

-- Represents the average level of complexity date:

$\overline{CM}$ - represents the average CM given by:

$$\overline{CM} = \frac{\sum_{i=1}^{NV} CM_i}{NV} \qquad (7)$$

S_{ICP} – is given by:

$$S_{ICP} = \sqrt{\frac{\sum_{i=1}^{NV} \left(ICP_i - \overline{ICP}\right)^2}{NV}} \qquad (8)$$

S_{CM} - is given by:

$$S_{CM} = \sqrt{\frac{\sum_{i=1}^{NV} \left(CM_i - \overline{CM}\right)^2}{NV}} \qquad (9)$$

For the program variants described in Table 13, rezults:

NV = 3

$$\overline{ICP} = \frac{0,46 + 0,30 + 0}{3} = 0,25 \qquad (10)$$

$$\overline{CM} = \frac{6 + 18 + 18}{3} = 14 \qquad (11)$$

	ICP - $\overline{ICP}$	(ICP - $\overline{ICP}$)^2	CH - $\overline{CH}$	(CH - $\overline{CH}$)^2
$PROG_1$	0,21	0,04	-8	64
$PROG_2$	0,05	0,00	4	16
$PROG_3$	-0,25	0,06	4	16

Table 14. Necessary values in computing the correlation between the correctness index and the cyclomatic complexity

$$S_{ICP} = \sqrt{\frac{0{,}04 + 0{,}01 + 0{,}06}{3}} = \sqrt{0{,}04} = 0{,}19 \quad (12)$$

$$S_{CM} = \sqrt{\frac{64 + 16 + 16}{3}} = \sqrt{32} = 5{,}66 \quad (13)$$

$$CC = \frac{0{,}21 \cdot (-8) + 0{,}05 \cdot 4 + (-0{,}25) \cdot 4}{3 \cdot 0{,}19 \cdot 5{,}66} = \frac{-2{,}47}{3{,}23} = = -0{,}76 \quad (14)$$

IV. Conclusion

As the coefficient of CF correlation is closest either -1 or 1 there is a strong link between the level of linear correctness and the McCabe complexity.

The Correlation stability on the results of the characteristics of quality is obtained through:

- Improving test procedures
- Enriching the list of programs under review
- Raising experience test team
- Increasing the diversity of spatial data test
- Implementation of the concept of completeness of the process of testing.

The use of quantitative methods for determining the intervals of confidence, too, stable is intended to create a new image on trust of users in the quality of results that the application informatics them to bring their disposal.

References

[1] Ivan Ion, Popescu Mihai - Metrici software, BYTE Romania, vol.2, nr.5, mai 1996, pg.73-82
[2] Ivan Ion, Teodorescu Laurențiu, Pocatilu Paul, Creşterea calității software prin testare, QMedia, Nr. 5, 2000
[3] Ivan Ion, Amancei Cristian, Stabilitatea coeficienților modelului global de calitate software, Editura ASE, Bucureşti, 2006
[4] Ion IVAN, Cătălin BOJA, Analiza metricilor software, Studii şi Cercetări de Calcul Economic şi Cibernetică Economică, vol. 40, nr. 1, 2006, pg.65- 78, ISSN 0585-7511
[5] Ion IVAN, Nicolae Iulian ENESCU, Stabilirea nivelului de corectitudine pentru aplicații informatice distribuite, Editura ASE, Bucureşti, 2009
[6] Nicolae Iulian ENESCU, "Modele pentru evaluarea corectitudinii aplicatiilor informatice distribuite", Teza de Doctorat, Bucuresti, 2008