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.

Still, there are errors on levels of aggregation ERR11, ERR12, ... ERRij, ... ERRNE.

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:

\[
ICP = \begin{cases} 
0, \text{ daca } \sum_{j=1}^{NT} \sum_{h=1}^{NE} PC_h \cdot ERR_{hj} = 0 \\
\sum_{j=1}^{NT} \sum_{h=1}^{NE} PC_h \cdot ERR_{hj}, \text{ else}
\end{cases}
\]

(1)
Complexity in McCabe way or Cyclomatic Complexity of a program is given in [4] by:

$$CM = na - nn + 2$$

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:

<table>
<thead>
<tr>
<th>Cyclomatic complexity</th>
<th>Risk evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-10</td>
<td>Simple program/module, low risk</td>
</tr>
<tr>
<td>11-20</td>
<td>Complex program/module, middle risk</td>
</tr>
<tr>
<td>21-50</td>
<td>Very complex program/module, high risk</td>
</tr>
<tr>
<td>Bigger than 50</td>
<td>Untestable program/module, very high risk</td>
</tr>
</tbody>
</table>

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:

$$1 \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}$$

where $a_{ij}$ are the integer coefficients of the A matrix, with $n$ lines and $m$ columns, and $a_{ij} \in [-100, 100]$, if $n \neq m$

$$\sum_{i=1}^{n} a_{ii}$$

$$\sum_{i=1}^{n} a_{i,n-i}$$

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

$$\prod_{j=0}^{k} v_{ij}$$

c) where $v_{ij}$ are integer elements of the V vector and $v_{ij} \in [-10, 10]$ 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.

<table>
<thead>
<tr>
<th>Test</th>
<th>Input data</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>$n = -1; m = 3$</td>
<td>Error: n outside the range</td>
</tr>
<tr>
<td>T12</td>
<td>$n = 2; m = 11$</td>
<td>Error: m outside the range</td>
</tr>
<tr>
<td>T13</td>
<td>$n = -2; m = 15$</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T14</td>
<td>$n = 3; m = 2$</td>
<td>Error: -101 outside the range</td>
</tr>
<tr>
<td>T15</td>
<td>$n = 2; m = 2$</td>
<td>Error: 102 outside the range</td>
</tr>
<tr>
<td>T16</td>
<td>$n = 3; m = 2$</td>
<td>Error: elements sum is zero</td>
</tr>
<tr>
<td>T17</td>
<td>$n = 3; m = 2$</td>
<td>Result is 0.20</td>
</tr>
</tbody>
</table>

Table 2. Test set for M1 function

Test set 3, SDT2, table 3.

<table>
<thead>
<tr>
<th>Test</th>
<th>Input data</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T21</td>
<td>$n = m = -1$</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T22</td>
<td>$n = m = 3$</td>
<td>Error: -101, 101 outside the range</td>
</tr>
<tr>
<td>T23</td>
<td>$n = m = 3$</td>
<td>Error: the sum of the elements from the secondary diagonal is 0</td>
</tr>
<tr>
<td>T24</td>
<td>$n = m = 3$</td>
<td>The result is 0.00</td>
</tr>
</tbody>
</table>
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];
    for(i = 0; i < n; i++)
        s2 += a[i][n-1-i];

    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{
        rez = M1(n, m);
    }
    printf("The result is 
", rez);
}
```
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.

<table>
<thead>
<tr>
<th>Test</th>
<th>Program result</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>Exception: Integer division by zero</td>
<td>Error: n outside the range</td>
</tr>
<tr>
<td>T12</td>
<td>Result is 0.00</td>
<td>Error: m outside the range</td>
</tr>
<tr>
<td>T13</td>
<td>Exception: Integer division by zero</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T14</td>
<td>Result is 0.00</td>
<td>Error: -101 outside the range</td>
</tr>
<tr>
<td>T15</td>
<td>Result is 37.00</td>
<td>Error: 102 outside the range</td>
</tr>
<tr>
<td>T16</td>
<td>Exception: Integer division by zero</td>
<td>Error: elements’ sum is 0</td>
</tr>
<tr>
<td>T17</td>
<td>Result is 0.00</td>
<td>The result is 0.20</td>
</tr>
<tr>
<td>T21</td>
<td>Exception: Integer division by zero</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T22</td>
<td>Result is 0.00</td>
<td>Error: -101. 101 outside the range</td>
</tr>
<tr>
<td>T23</td>
<td>Exception: Integer division by zero</td>
<td>Error: the sum of the elements from the secondary diagonal is 0</td>
</tr>
<tr>
<td>T24</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T25</td>
<td>Result is 0.00</td>
<td>Result is 0.80</td>
</tr>
<tr>
<td>T31</td>
<td>Result is 1.00</td>
<td>Error: n or m outside the range</td>
</tr>
<tr>
<td>T32</td>
<td>Exception: Access violation reading location 0x0000000b</td>
<td>Error: m or n outside the range</td>
</tr>
<tr>
<td>T33</td>
<td>Result is 0.00</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>T34</td>
<td>Exception: Integer division by zero</td>
<td>Error: elements’ product is 0</td>
</tr>
<tr>
<td>T35</td>
<td>Result is 0.00</td>
<td>Result is -0.16</td>
</tr>
<tr>
<td>T41</td>
<td>Exception: Integer division by zero</td>
<td>Error: c is 0</td>
</tr>
<tr>
<td>T42</td>
<td>Result is 0.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T43</td>
<td>Result is -1.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T44</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T45</td>
<td>Result is 1.00</td>
<td>Result is 1.00</td>
</tr>
</tbody>
</table>

Table 6. Running time result

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

- Exceptions, noted with E1, for which a 0.7 gravity coefficient is assigned
- Wrong results, noted with E2, for which a 0.3 gravity coefficient is assigned

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

<table>
<thead>
<tr>
<th>Test set</th>
<th>E1</th>
<th>E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>SDT2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SDT3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>SDT4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 7. Error number found for the test sets

The correctness indicator, ICP1, for the PROG1 program is

\[
ICP_1 = \frac{0.7 \times 8 + 0.3 \times 11}{8 + 11} = \frac{5.6 + 3.3}{19} = 0.46
\]

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

![Figure 1. The graph structure for the PROG1](image)

For M1 function rezult the graph structure from figure 2:

![Figure 2. The graph structure for M1 module](image)

For M2 function rezult the graph structure from figure 3:

![Figure 3. The graph structure for M2 module](image)
Figure 3. The graph structure for M2 module
For M3 function result the graph structure from figure 4:

Figure 4. The graph structure for M3 module
For M4 function result 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.

<table>
<thead>
<tr>
<th></th>
<th>na</th>
<th>nn</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>10</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>M2</td>
<td>15</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>M3</td>
<td>7</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>M4</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>main</td>
<td>13</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>PROG1</td>
<td>47</td>
<td>43</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 8. McCabe complexity for PROG1

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

```c
#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\n");
        } else if (error == 100)
        {
            printf("Error: m outside the range\n");
        } else
        {
            printf("Error: n outside the range\n");
        }
        return -1000;
    }
    printf("Elements of A matrix:\n");
    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\n");
        return -1000;
    }
    if (s == 0)
    {
        printf("Error: elements' sum is 0\n");
        return -1000;
    }
    return 1/s;
}
```

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

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;

void main(){
    int n, m;
    float rez = 0;
    printf("n=");
    scanf("%d", &n);
    printf("m=");
    scanf("%d", &m);
    if (n == 1)
        rez = M4();
    else
        rez = M2(n);

    if (rez != -1000)
        printf("The result is %f", rez);
}

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++;
        }
    }
    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);
}

float M2(int n){
    int 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;
    }
    return (s1/s2);
After runtime, with the input data from 2, 3, 4 and 5 tables, the table 9 is obtained.

<table>
<thead>
<tr>
<th>Test</th>
<th>Program’s result</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>Error: n outside the range</td>
<td>Error: n outside the range</td>
</tr>
<tr>
<td>T12</td>
<td>Error: m outside the range</td>
<td>Error: m outside the range</td>
</tr>
<tr>
<td>T13</td>
<td>Error: n and m outside the range</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T14</td>
<td>Error: -101 outside the range</td>
<td>Error: -101 outside the range</td>
</tr>
<tr>
<td>T15</td>
<td>Error: 102 outside the range</td>
<td>Error: 102 outside the range</td>
</tr>
<tr>
<td>T16</td>
<td>Error: elements’ sum is 0</td>
<td>Error: elements’ sum is 0</td>
</tr>
<tr>
<td>T17</td>
<td>Result is 0.00</td>
<td>Result is 0.20</td>
</tr>
<tr>
<td>T18</td>
<td>Error: n and m outside the range</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T19</td>
<td>Error: -101 and 101 outside the range</td>
<td>Error: -101 and 101 outside the range</td>
</tr>
<tr>
<td>T20</td>
<td>Error: the sum of the elements from the secondary diagonal is 0</td>
<td>Error: the sum of the elements from the secondary diagonal is 0</td>
</tr>
<tr>
<td>T21</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T22</td>
<td>Error: n or m outside the range</td>
<td>Error: n or m outside the range</td>
</tr>
<tr>
<td>T23</td>
<td>Error: 11 outside the range</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>T24</td>
<td>Error: elements’ product is 0</td>
<td>Error: elements’ product is 0</td>
</tr>
<tr>
<td>T25</td>
<td>Result is 0.00</td>
<td>Result is -0.16</td>
</tr>
<tr>
<td>T26</td>
<td>Error: c is 0</td>
<td>Error: c is 0</td>
</tr>
<tr>
<td>T27</td>
<td>Result is 0.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T28</td>
<td>Error: 11 outside the range</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>T29</td>
<td>Error: elements’ product is 0</td>
<td>Error: elements’ product is 0</td>
</tr>
<tr>
<td>T30</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T31</td>
<td>Error: c is 0</td>
<td>Error: c is 0</td>
</tr>
<tr>
<td>T32</td>
<td>Result is 0.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T33</td>
<td>Error: 11 outside the range</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>T34</td>
<td>Error: elements’ product is 0</td>
<td>Error: elements’ product is 0</td>
</tr>
<tr>
<td>T35</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T36</td>
<td>Error: c is 0</td>
<td>Error: c is 0</td>
</tr>
<tr>
<td>T37</td>
<td>Result is 0.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T38</td>
<td>Error: 11 outside the range</td>
<td>Error: 11 outside the range</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Test set</th>
<th>E1</th>
<th>E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SD2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SD3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

The correctness indicator, \( ICP_2 \), for PROG2 program is:

\[
ICP_2 = \frac{0.7 \times 0 + 0.3 \times 6}{6} = \frac{1.8}{6} = 0.30
\]  

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

![Figure 6. The graph structure for the PROG2](image)

For M1 function result the graph structure from figure 7:
Figure 7. The graph structure for M1 module

For M2 function result the graph structure from figure 8:

Figure 8. The graph structure for M2 module

For M3 function result the graph structure from figure 9:

Figure 9. The graph structure for M3 module
For M4 function result the graph structure from figure 10:

![Graph Structure](image)

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.

<table>
<thead>
<tr>
<th></th>
<th>na</th>
<th>nm</th>
<th>CM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>43</td>
<td>36</td>
<td>9</td>
</tr>
<tr>
<td>M2</td>
<td>34</td>
<td>30</td>
<td>6</td>
</tr>
<tr>
<td>M3</td>
<td>26</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>M4</td>
<td>8</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>main</td>
<td>15</td>
<td>12</td>
<td>5</td>
</tr>
<tr>
<td>PROG2</td>
<td>126</td>
<td>110</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 11. McCabe complexity for PROG2

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

```c
#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;
}

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++){
            scanf("%d", &a[i][j]);
            if (a[i][j] < 0 || 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;
    }
    if (s1 == 0){
        printf("Error: elements' sum is 0");
        return -1000;
    }
    return 1.0/s;
}
```

71
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], i, 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] < D10 || 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 == 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.

<table>
<thead>
<tr>
<th>Test</th>
<th>Program’s result</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>Error: n out of range</td>
<td>Error: n out of range</td>
</tr>
<tr>
<td>T12</td>
<td>Error: m out of range</td>
<td>Error: m out of range</td>
</tr>
<tr>
<td>T13</td>
<td>Error: n and m out of range</td>
<td>Error: n and m out of range</td>
</tr>
<tr>
<td>T14</td>
<td>Error: -101 out of range</td>
<td>Error: -101 out of range</td>
</tr>
<tr>
<td>T15</td>
<td>Error: 102 out of range</td>
<td>Error: 102 out of range</td>
</tr>
<tr>
<td>T16</td>
<td>Error: elements’ sum is 0</td>
<td>Error: elements’ sum is 0</td>
</tr>
<tr>
<td>T17</td>
<td>Result is 0.20</td>
<td>Result is 0.20</td>
</tr>
<tr>
<td>T21</td>
<td>Error: n and m out of range</td>
<td>Error: n and m out of range</td>
</tr>
</tbody>
</table>
It can be observed that after running the tests, there are no more errors. The correctness indicator, ICP, for PROG3 program is 0. CM for PROG3 is identical with CM for PROG2.

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

<table>
<thead>
<tr>
<th></th>
<th>Index of correctness (ICP)</th>
<th>McCabe Complexity (CM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG1</td>
<td>0.46</td>
<td>6</td>
</tr>
<tr>
<td>PROG2</td>
<td>0.30</td>
<td>18</td>
</tr>
<tr>
<td>PROG3</td>
<td>0</td>
<td>18</td>
</tr>
</tbody>
</table>

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

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

\[
CC = \frac{\sum_{i=1}^{NV} \left( ICP_i - \bar{ICP} \right) \left( CM_i - \bar{CM} \right)}{\sum_{i=1}^{NV} S_{ICP_i} \cdot S_{CM_i}}
\]  

(5)

where:

- NV - represents the number of variations of the program -- The index represents the average accuracy given by:

\[
\bar{ICP} = \frac{\sum_{i=1}^{NV} ICP_i}{NV}
\]  

(6)

-- Represents the average level of complexity date:

\[
\bar{CM} = \frac{\sum_{i=1}^{NV} CM_i}{NV}
\]  

(7)

S_{ICP} – is given by:

\[
S_{ICP} = \sqrt{\frac{\sum_{i=1}^{NV} (ICP_i - \bar{ICP})^2}{NV}}
\]  

(8)

S_{CM} – is given by:

\[
S_{CM} = \sqrt{\frac{\sum_{i=1}^{NV} (CM_i - \bar{CM})^2}{NV}}
\]  

(9)

For the program variants described in Table 13, results:

\[
NV = 3
\]

\[
\bar{ICP} = \frac{0.46 + 0.30 + 0}{3} = 0.25
\]  

(10)

\[
\bar{CM} = \frac{6 + 18 + 18}{3} = 14
\]  

(11)

<table>
<thead>
<tr>
<th></th>
<th>ICP</th>
<th>CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG1</td>
<td>0.21</td>
<td>-8</td>
</tr>
<tr>
<td>PROG2</td>
<td>0.05</td>
<td>4</td>
</tr>
<tr>
<td>PROG3</td>
<td>-0.25</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 14. Necessary values in computing the correlation between the correctness index and the cyclomatic complexity
\[ S_{K_P} = \sqrt{\frac{0.04 + 0.01 + 0.06}{3}} = \sqrt{0.04} = 0.19 \]  
(12)

\[ S_{CM} = \sqrt{\frac{64 + 16 + 16}{3}} = \sqrt{32} = 5.66 \]  
(13)

\[ CC = \frac{0.21 \cdot (-8) + 0.05 \cdot 4 + (-0.25) \cdot 4}{3 \cdot 0.19 \cdot 5.66} = \frac{-2.47}{3.23} = -0.76 \]  
(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

[2] Ivan Ion, Teodorescu Laurențiu, Pocatilu Paul, Creșterea calității software prin testare, QMedia, Nr. 5, 2000