Increasing Level of Correctness in Correlation with and Reliability Level

DAN MANCAS
NICOLAE-IULIAN ENESCU
ECATERINA - IRINA MANOLE
Computer and Communication Engineering Department
Faculty of Automation, Computers and Electronics
Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj
ROMANIA
dan.mancas@ucv.ro, nenescu@cs.ucv.ro, catya_ace@yahoo.com

Abstract: - The scope of our research is finding a correlation between the correctness indicator and the reliability indicator for software programs. For this, the correctness and reliability 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 reliability in the way that for an increasing of the correctness level there will also be a significant increase of the reliability level.

Key-Words: - testing, correctness, reliability, correlation

1 Informatics 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.
Reliability is the most important software characteristic and relates to the capacity of an informatic application to maintain its performance level in certain given conditions for a specified period of time. A software product has to have as less as possible failures and to be easy to repair.
Correctness is the quality characteristic that is the hardest to obtain. For software products with a high complexity, a complete testing is imposible to be made, in order to offer the assurance that all error have been found and corrected.
Application correctness are parted in four categories:
- Sintactical correctness which presumes that the program is correct when it compiles without errors; in other words, during runtime; when a program is sintactically 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.

2 Defining used formulas
Correctness of an application is marked out in testing.
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 NT.

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

It defines indicator of correctness by ICP relationship:

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

Reliability is seen as a trust measure for the software’s capacity to function properly in the initial conditions. The definition is related to the probability that an error can be activated by a specific input data set in a component module of a program.

The reliability of the program NIVF comes from the relationship:

\[
NIVF = \frac{k}{n}
\]

where:

- \(n\) - represents the total number of tests
- \(k\) - the number of tests performed successfully

### 3 Problem Solution

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

\[
\frac{1}{\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}}
\]

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 \(n = m\)

\[
\frac{1}{\prod_{i=0}^{k} v_i}
\]

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 \(n=0\) 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 1.

<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>(A = \begin{pmatrix} 1 &amp; 10 \ 0 &amp; -101 \ 1 &amp; 2 \end{pmatrix})</td>
<td>Error: 102 outside the range</td>
</tr>
<tr>
<td>T16</td>
<td>n = 3; m = 2</td>
<td>Error:</td>
</tr>
</tbody>
</table>
$A = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$

elements sum is zero

$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$

Result is 0.20

Test set 2, SDT₂, table 2.

Table 2. Test set for M² function

<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>A = \begin{pmatrix} 10 &amp; 4 &amp; 5 \ 6 &amp; -9 &amp; -101 \ 101 &amp; 8 &amp; 7 \end{pmatrix}</td>
<td></td>
<td></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>A = \begin{pmatrix} 1 &amp; 8 &amp; -9 \ 1 &amp; -1 &amp; 5 \ 10 &amp; 9 &amp; 5 \end{pmatrix}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₂₄</td>
<td>n = m = 3</td>
<td>The result is 0.00</td>
</tr>
<tr>
<td>A = \begin{pmatrix} -1 &amp; 8 &amp; 10 \ 9 &amp; 1 &amp; 5 \ 0 &amp; 8 &amp; 0 \end{pmatrix}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₂₅</td>
<td>n = m = 3</td>
<td>The result is 0.80</td>
</tr>
<tr>
<td>A = \begin{pmatrix} 1 &amp; 2 &amp; 3 \ 4 &amp; 5 &amp; 6 \ 7 &amp; 8 &amp; 6 \end{pmatrix}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test set 3, SDT₃, table 3.

Table 3. Test set for M³ function

<table>
<thead>
<tr>
<th>Test</th>
<th>Input data</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₃₁</td>
<td>n = 0; m = -1</td>
<td>Error: m outside the range</td>
</tr>
<tr>
<td>T₃₂</td>
<td>n = 11; m = 0</td>
<td>Error: n outside the range</td>
</tr>
<tr>
<td>T₃₃</td>
<td>n = 0; m = 3</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>V = \begin{pmatrix} 11 &amp; 1 &amp; -1 \end{pmatrix}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₃₄</td>
<td>n = 0; m = 3</td>
<td>Error: elements product is 0</td>
</tr>
<tr>
<td>V = \begin{pmatrix} 0 &amp; 5 &amp; -1 \end{pmatrix}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₃₅</td>
<td>n = 3; m = 0</td>
<td>The result is -0.16</td>
</tr>
</tbody>
</table>

$V = \begin{pmatrix} 1 & -3 & 2 \end{pmatrix}$

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]);
            s₁ += a[i][j];
            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 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 1, 2, 3 and 4 the table 5 is obtained.

Table 5. Running time result

<table>
<thead>
<tr>
<th>Test</th>
<th>Program result</th>
<th>Expected result</th>
</tr>
</thead>
<tbody>
<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>T18</td>
<td>Exception: Integer division by zero</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T19</td>
<td>Result is 0.00</td>
<td>Error: -101. 101 outside the range</td>
</tr>
<tr>
<td>T20</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>T21</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T22</td>
<td>Result is 0.00</td>
<td>Result is 0.80</td>
</tr>
<tr>
<td>T23</td>
<td>Exception: Access violation reading location 0x0000000b</td>
<td>Error: m or n outside the range</td>
</tr>
<tr>
<td>T24</td>
<td>Result is 0.00</td>
<td>Error: 11 outside the range</td>
</tr>
<tr>
<td>T25</td>
<td>Result is 0.00</td>
<td>Error: elements’ product is 0</td>
</tr>
<tr>
<td>T26</td>
<td>Result is 0.00</td>
<td>Result is -0.16</td>
</tr>
<tr>
<td>T27</td>
<td>Exception: Integer division by zero</td>
<td>Error: c is 0</td>
</tr>
<tr>
<td>T28</td>
<td>Result is 0.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T29</td>
<td>Result is -1.00</td>
<td>Error: -b/c is less than 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>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>T36</td>
<td>Exception: Integer division by zero</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>Result is -1.00</td>
<td>Error: -b/c is less than 0</td>
</tr>
<tr>
<td>T39</td>
<td>Result is 0.00</td>
<td>Result is 0.00</td>
</tr>
<tr>
<td>T40</td>
<td>Result is 1.00</td>
<td>Result is 1.00</td>
</tr>
</tbody>
</table>

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 6 is obtained:

Table 6. Error number found for the test sets
<table>
<thead>
<tr>
<th>Test set</th>
<th>$E_1$</th>
<th>$E_2$</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><strong>Total</strong></td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

The correctness indicator, $ICP_1$, for the PROG1 program is

$$ICP_1 = \frac{0.7 \times 8 + 0.3 \times 11}{8 + 11} = \frac{5.6 + 3.3}{19} = 0.46$$ \hspace{1cm} (3)

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

```c
#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: n outside the range ");
        else{
            printf("Error: m outside the range ");
        }
    }
    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;
}
```

```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 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];
    }
    s1 += s2;
    return 1/s;
}
```
```c
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;
    }
    return sqrt(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)
    {
        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 1, 2, 3 and 4 tables, the table 7 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>
</tbody>
</table>
```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>T21</td>
<td>Error: n and m outside the range</td>
<td>Error: n and m outside the range</td>
</tr>
<tr>
<td>T22</td>
<td>Error: -101 and 101 outside the range</td>
<td>Error: -101 and 101 outside the range</td>
</tr>
<tr>
<td>T23</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>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>Error: n or m outside the range</td>
<td>Error: n or m outside the range</td>
</tr>
<tr>
<td>T32</td>
<td>Error: n or m outside the range</td>
<td>Error: n or m outside the range</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 -0.16</td>
</tr>
<tr>
<td>T41</td>
<td>Error: c is 0</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 0.00</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>

The correctness indicator, ICP₂, for PROG₂ program is:

\[
ICP₂ = \frac{0.7 \times 0 + 0.3 \times 6}{6} = \frac{1.8}{6} = 0.30
\]

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 > 100)
        error = 100;
    else if (error == 100)
        printf("Error: n is outside the range ");
    else{
        printf("Error: n is outside the range ");
    }
    if (m <= 0 || m > 100)
        error ++ = 100;
    else{
        printf("Error: m is outside the range ");
    }
    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:");
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++;
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++;
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++;
}
```

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 8.

**Table 8. Errors found for the PROG2 with the given data sets**

<table>
<thead>
<tr>
<th>Test set</th>
<th>E₁</th>
<th>E₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT₁</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SDT₂</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SDT₃</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The correctness indicator, ICP₂, for PROG₂ program is:

\[
ICP₂ = \frac{0.7 \times 0 + 0.3 \times 6}{6} = \frac{1.8}{6} = 0.30
\]

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 > 100)
        error = 100;
    else if (error == 100)
        printf("Error: n is outside the range ");
    else{
        printf("Error: n is outside the range ");
    }
    if (m <= 0 || m > 100)
        error ++ = 100;
    else{
        printf("Error: m is outside the range ");
    }
    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:");
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++;
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++;
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++;
}
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SDT₄</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
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("elements’ sum is 0");
    return -1000;
}
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        if (a[i][j] < -10 || a[i][j] > 10)
            err[l] = a[i][j];
        else
            s1 += a[i][j];
        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++)
        if (c == 0)
            printf("Error: c is 0");
        return -1000;
    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=", &n);
    scanf("%d", &n);
    printf("m=", &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 9.

Table 9. PROG3 program result after runtime

<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>
<tr>
<td>T22</td>
<td>Error: -101, 101 out of range</td>
<td>Error: -101, 101 out of range</td>
</tr>
<tr>
<td>T23</td>
<td>Error: elements’ sum from the secondary diagonal is 0</td>
<td>Error: elements’ sum from the secondary diagonal is 0</td>
</tr>
</tbody>
</table>

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

The correctness indicator, ICP3, for PROG3 program is 0.

For datasets test SDT1, SDT2, SDT3 and SDT4 after the execution of the program is obtained PROG1 level of reliability NIVF1 as:

\[ NIVF_1 = \frac{3}{22} = 0.14 \] (4)

After the execution of the program is obtained PROG2 level of reliability NIVF2 as:

\[ NIVF_2 = \frac{17}{22} = 0.85 \] (5)

After the execution of the program is obtained PROG3 level of reliability NIVF3 as:

\[ NIVF_3 = \frac{22}{22} = 1 \] (6)

So the level of correctness and reliability programs PROG1, PROG2 and PROG3 is presented in Table 10 and Figure 1.

Table 10. The level of accuracy and reliability programs PROG1, PROG2 and PROG3

<table>
<thead>
<tr>
<th>Program</th>
<th>Index of correctness (ICP)</th>
<th>Level of reliability (NIVF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROG1</td>
<td>0.46</td>
<td>0.14</td>
</tr>
<tr>
<td>PROG2</td>
<td>0.30</td>
<td>0.85</td>
</tr>
<tr>
<td>PROG3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The coefficient of correlation between the level of accuracy and reliability $CF$ is calculated by the formula:

$$CF = \frac{\sum_{i=1}^{NV} (ICP_i - \overline{ICP}) \cdot (NIVF_i - \overline{NIVF})}{NV \cdot S_{ICP} \cdot S_{NIVF}}$$

where:

$NV$ - represents the number of variations of the program

-- The index represents the average accuracy given by:

$$\overline{ICP} = \frac{\sum_{i=1}^{NV} ICP_i}{NV}$$

-- Represents the average level of reliability date:

$$\overline{NIVF} = \frac{\sum_{i=1}^{NV} NIVF_i}{NV}$$

$S_{ICP}$ - comes from:

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

$S_{NIVF}$ - comes from:

$$S_{NIVF} = \sqrt{\frac{\sum_{i=1}^{NV} (NIVF_i - \overline{NIVF})^2}{NV}}$$

For variations on the program described in the table 11, result:
results that the application informatics distributed them to bring their disposal.
Software reliability is negatively influenced by:
- Absence of input data validation
- Absence of intermediary results tests
- Absence of testing the denominator before a division.
Accordingly, in order to increase the reliability, there are searched ways to counteract the negative factors apparition:
- Providing powerful libraries for validation procedures
- Constraining the denominator tests before any division
- Constraining control variables membership validation at the interval defined in the program

References: