

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 impossible to be made, in order to offer the assurance that all errors have been found and corrected.

Application correctness are 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 code 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.

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:

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

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:

$$NIVF = \frac{k}{n} \quad (2)$$

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:

- a) $\frac{1}{\sum_{i=1}^n \sum_{j=1}^m a_{ij}}$ where a_{ij} are the integer coefficients of the A matrix, with n lines and m columns, and $a_{ij} \in [-100, 100]$, if $n \neq m$
- b) $\frac{\sum_{i=1}^n a_{ii}}{\sum_{i=1}^n a_{i,n-i}}$ where a_{ij} are the integer coefficients of the square A matrix and $a_{ij} \in [-10, 10]$, if $n = m$
- c) $\frac{1}{\prod_{i=0}^k v_i}$ 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 $n=0$, 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 1. Test set for M1 function

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	Error:

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

Test set 2, SDT₂, table 2.

Table 2. Test set for M2 function

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}	$n = m = 3$ $A = \begin{pmatrix} -1 & 8 & 10 \\ 9 & 1 & 5 \\ 0 & 8 & 0 \end{pmatrix}$	The result is 0.00
T_{25}	$n = m = 3$ $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \end{pmatrix}$	The result is 0.80

Test set 3, SDT₃, table 3.

Table 3. Test set for M3 function

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}	$n = 3; m = 0$	The result is -0.16

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

Test set 4, SDT₄, table 4.

Table 4. Test set for M4 function

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

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 1, 2, 3 and 4 the table 5 is obtained.

Table 5. Running time result

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

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

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

The correctness indicator, ICP₁, for the PROG1 program is

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

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 1, 2, 3 and 4 tables, the table 7 is obtained.

Table 7. Runtime result for PROG2

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	Error: 102 outside the range

	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

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

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

The correctness indicator, ICP₂, for PROG2 program is:

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

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 9.

Table 9. PROG₃ program result after runtime

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 out of range	Error: -101, 101 out 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

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:

$$NIVF_1 = \frac{3}{22} = 0,14 \quad (4)$$

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

$$NIVF_2 = \frac{17}{22} = 0,85 \quad (5)$$

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

$$NIVF_3 = \frac{22}{22} = 1 \quad (6)$$

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

	Index of correctness (ICP)	Level of reliability (NIVF)
PROG ₁	0,46	0,14
PROG ₂	0,30	0,85
PROG ₃	0	1

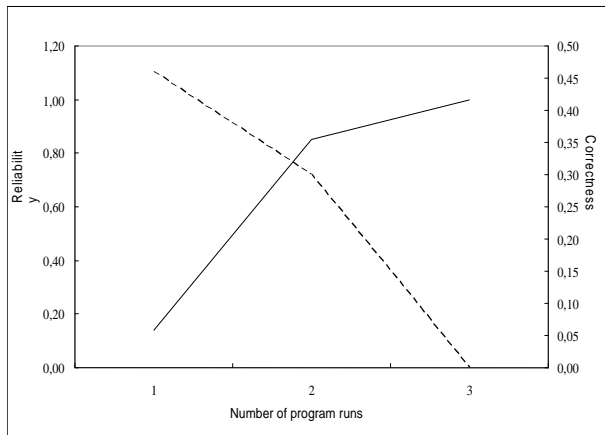


Figure 1. The relationship between correctness and reliability

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:

$$NV = 3$$

$$\overline{ICP} = \frac{0,46 + 0,30 + 0}{3} = 0,25$$

$$\overline{NIVF} = \frac{0,14 + 0,85 + 1}{3} = 0,66$$

Table 11. Values necessary mathematical correlation coefficient between the index level of correctness and reliability

	ICP	NIVF	$ICP - \overline{ICP}$
PROG ₁	0,46	0,14	0,21
PROG ₂	0,30	0,85	0,05
PROG ₃	0	1	-0,25

	$(ICP - \overline{ICP})^2$	$(NIVF - \overline{NIVF})^2$	$(NIVF - \overline{NIVF})^2$
PROG ₁	0,04	-0,52	0,27
PROG ₂	0,00	0,19	0,03
PROG ₃	0,06	0,34	0,11

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

$$S_{NIVF} = \sqrt{\frac{0,27 + 0,03 + 0,11}{3}} = \sqrt{0,14} = 0,38$$

$$CF = \frac{0,21 \cdot (-0,52) + 0,05 \cdot 0,19 + (-0,25) \cdot 0,34}{3 \cdot 0,19 \cdot 0,38} = \frac{-0,18}{0,21} = -0,86$$

4 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 reliability.

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 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:

- [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, *Creșterea calității software prin testare*, QMedia, Nr. 5, 2000
- [3] Ivan Ion, Amancei Cristian, *Stabilitatea coeficienților modelului global de calitate software*, Editura ASE, București, 2006
- [4] Ion IVAN, Cătălin BOJA, *Analiza metricilor software*, Studii și Cercetări de Calcul Economic și 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, București, 2008
- [6] Nicolae Iulian ENESCU, “*Modele pentru evaluarea corectitudinii aplicațiilor informatice distribuite*”, Teza de Doctorat, Bucuresti, 2008