JOURNAL OF APPLIED QUANTITATIVE METHODS

Reliability and Quality Control – Practice and Experience

SOFTWARE RELIABILITY FROM THE FUNCTIONAL TESTING PERSPECTIVE

Mihai POPESCU
PhD, Senior Lecturer
Military Technical Academy, Bucharest, Romania

E-mail: popescum@mta.ro

Abstract: *The metrics proposed in this paper give a methodological framework in the field of the functional testing for the software programs.*
The probability of failure for software depends of the number of residual defects; obvious the detection of these depends very much of data test used, that are conforming with a operational profile.
But it's the same true that a linear source code, that have a lot of instructions, but a sequential structure, is easier tested and debugged than a code with alternative control structures.

Key words: *software; reliability; functional testing; metrics*

1. A metric of the complexity

There are very much specialists in the field that say the probability of failure for software depends of the number of residual defects; obvious the detection of these depends very much of data test used, that are conforming with a operational profile.

But it's the same true that a linear source code, that have a lot of instructions, but a sequential structure, is easier tested and debugged than a code with alternative control structures.

That explains why I defined in [POPE02][1] failure aprioristic probabilities (p_k) of software modules across with theirs cyclomatic complexity.

The choosing of the computing way for the probability p_k is determined by the available data and by the estimation and prediction models for reliability.

A usual weight computing formulas for failure aprioristic probabilities of the modules is (1):

$$p_k = \frac{\lambda_k}{\sum_{k=1}^{M} \lambda_k} \qquad (1)$$

where λ_k=failure intensity of module k, that is calculated by formulas:

$$\lambda_k = r * Me * \omega_k / I_k \qquad (2)$$

Where: Me=4,2*10^{-7} Musa rate for failures exposure;

r = processor speed (istructions/s) – can be established from benchmarking programs or from the technical characteristics given by the sale man;

ω_k = number of failures contained by the software module k. It can be determined in according to [ROME97], transforming source instructions written in a program language in function points and than determining the number of failures in according to CMM level (***C**apability **M**aturity **M**odel*) selected;

I_k = number of executable code lines k * expanded rate [ROME97].

This paragraph wants to deduct new metrics for complexity and reliability based on functional theory. For this goal, we'll note with:

T_i=duration of test i;

θ_i= average duration for locating/recovery of the module j .

Appropriate, we'll define the duration for locating/recovery of the a failure module being a random discrete variable, called T_{loc}, having the next repartition law :

$T_{loc} : \begin{pmatrix} Tloc_k \\ p_k \end{pmatrix}$, where:

$Tloc_k$= duration for locating/recovery cumulated on the branch k of the tree associated to the program P;

P_k= failure aprioristic probability of the module k.

For the locating tree of failure modules from fig.2, obtained based on the program structure proposed in fig. 1, we'll have the next repartition law for the random discrete variable T_{loc} :

$$T_{loc} = \begin{vmatrix} T_4+T_5 & T_4+T_5 & T_4+T_5 & T_4+T_5 & T_4+T_5 \\ +T_2+\theta_1 & +\theta_2 & +T_2+\theta_3 & +\theta_4 & +\theta_5 \\ p_1 & p_2 & p_3 & p_4 & p_5 \end{vmatrix}$$

Appropriate, we'll have the average duration of locating/recovery for module T_{loc_med}, the next expression :

$$T_{loc\_med} = (T_4+T_5+T_2+\theta_1)p_1 + (T_4+T_5+\theta_2)p_2+(T_4+T_5+T_2+\theta_3)p_3 + +(T_4+T_5+\theta_4)p_4+(T_4+T_5+\theta_5)p_5 \qquad (3)$$

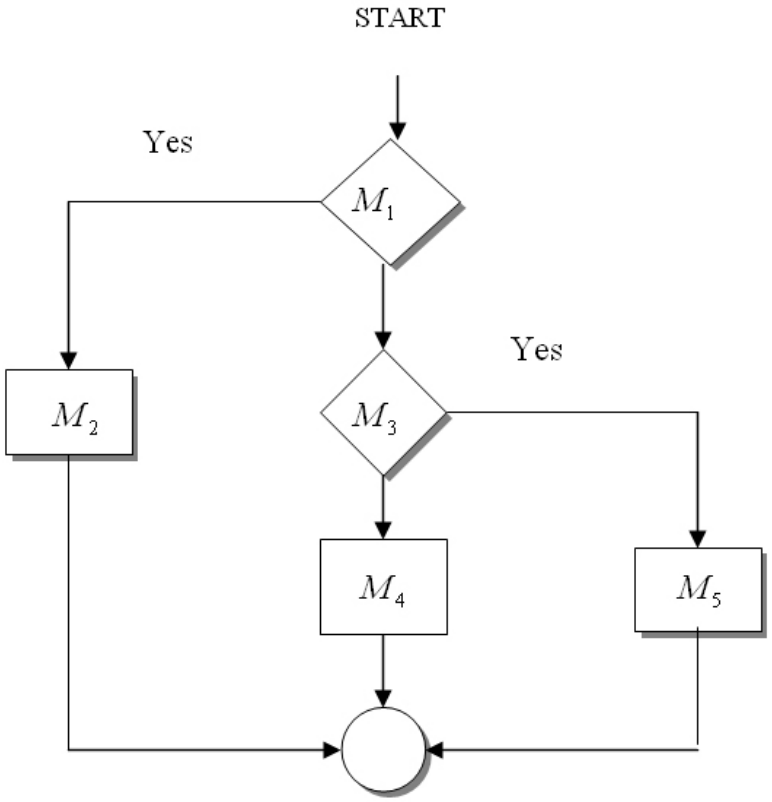

Figure 1. The decomposition of a program in modules for testing

We define variable Π_k like:

$$\Pi_k = \frac{T_{loc_k} \cdot p_k}{T_{loc\_med}}; \tag{4}$$

It can observe that Π_k has the next properties:

1) $\Pi_k > 0$;

2) $\sum_{k=1}^{N} \Pi_k = 1$,

that means the weight of the module k in the testing and recovery of module k,versus of the N program modules.

Fixing a duration for locating/recovery, $T^0_{loc\_med,}$ to accomplish a mission by software, the program must be written to satisfy the condition:

$$T_{loc\_med} \le T^0_{loc\_med}.$$

We can observe, in the same time, that descended sorting Π_k values on can see the modules with a big duration for testing/locating of the singular failures, these modules could be redesigned, eventually.

If we call E the number of residual defects from program, than:

$T_{loc\_gen\_med}=E\ \ T_{toc\_med}$. (5)

If $T_i=1,\ \forall i$ and

$\theta_j=1,\ \forall j$, we have:

$T_{loc\_med}=M_{loc\_med}$, and we'll get the number of steps to locate failure modules.

The using of these metrics has the next advantages:

- gives a better reflection of the locating and fixing mechanism for the defects contained in software (based on functional testing);
- gives value to the performances of the testing tools and to the skill of the debugging personal;
- offers a good support to software products designers, signaling the modules that need a bigger testing/locating time, suggesting even theirs redesign.

2. Case study

I'll compute the average duration of locating (3) and general average duration of locating (5) based on the methodology proposed at 1. and on the structure of the modules tree from fig. 1.

According to this structure, we'll start from the next information that we know about the modules and functions (table 2).

Table 2. Information known about software modules

Module Name	Functions	Number of executable instructions function/module	Probability of execution function/module	Programming Language
M1	—	3	1	C++
M2	push(z); z ∈ [1,50]	15	1	C++
M3	—	4	1	C++
M4	push(z),top(z), pop(z); z ∈ [1,50]	push(z) - 15 pop() - 10 top(z) - 5	pop() – 0.8 top(z) – 0.1 push(z) – 0.1	C++ C++ C++
M5	push(z),pop(), top(z);z ∈ [1,50]	push(z) - 15 pop() - 10 top(z) - 5	push(z) – 0.8 top(z) – 0.1 pop() – 0.1	C++ C++ C++

For this goal, we proceed the next steps:

1°)The calculation of the modules' failure aprioristic probabilities with (3) and (4) formulas.

We admit the hypothesis that software will be executed on a 2 MIPS computer, meaning r=2000000, and CMM level is 3, a common used level for IT companies [PAUL93].

With the explanations given for (2) and with information obtained from [POPE02], we'll have:

$\omega_1 =$ 3 : 53 * 1.63 = 0.923 defects;

$\omega_2 =$ 15 : 53 * 1.63 = 0.461 defects;

$\omega_3 =$ 4 : 53 * 1.63 = 0.123 defects;

$\omega_4 =$ (15+10+5) : 53 * 1.63 = 0.923 defects;

$\omega_5 =$ (15+10+5) : 53 * 1.63 = 0.923 defects;

$I_1 =$ (3 source lines) * (object instructions/source line) = 18 object instructions;

$I_2 =$ (15 source lines) * (6 object instructions/source line) = 90 object instructions;

$I_3 =$ (4 source lines) * (6 object instructions/source line) = 24 object instructions;

$I_4 =$ (30 source lines) * (6 object instructions/source line) = 180 object instructions;

$I_5 =$ (30 source lines) * (6 object instructions/source line) = 180 object instructions;

Replacing these values, we'll have:

$$\lambda_1 = r * Me * \omega_1 / I_1 = \frac{2000000 instructions}{\sec ond} * \left(4.2 \cdot 10^{-7} \frac{failures}{defect}\right) * 0.092 defects$$

/18 = 0.0042 $\frac{defects}{\sec ond}$

$$\lambda_2 = 2000000 * (4.3 * 10 - 7) * 0.123/24 = 0.0044 \frac{defects}{\sec ond};$$

$$\lambda_3 = 2000000 * (4.2 \cdot 10 - 7) * 0.123/24 = 0.044 \frac{defects}{\sec ond};$$

$$\lambda_4 = 2000000 * (4.2 \cdot 10 - 7) * 0.923/180 = 0.045 \frac{defects}{\sec ond};$$

$$\lambda_5 = 2000000 * (4.2 \cdot 10 - 7) * 0.923/180 = 0.045 \frac{defects}{\sec ond};$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 = 0.0219 \frac{defects}{\sec ond};$$

The value a little big for the failure intensity is explained by the big number of instructions contained by modules and by r value(2 MIPS), big enough.

Applying formula(1), we have:

$$p_1 = \lambda_1 \Big/ \sum_{i=1}^{5} \lambda_i = 0.0042/0.0219 = 0.193;$$

$$p_2 = \lambda_2 / 0.0219 = 0.0043/0.0219 = 0.196;$$

$$p_3 = \lambda_3 / 0.0219 = 0.0044/0.0219 = 0.201;$$

$$p_4 = \lambda_4 / 0.0219 = 0.0045/0.0219 = 0.205;$$

$$p_5 = \lambda_5 / 0.0219 = 0.0045/0.0219 = 0.205.$$

According to the values of these probabilities, FIAB program [GHIT96] displays failure tree from fig. 2.

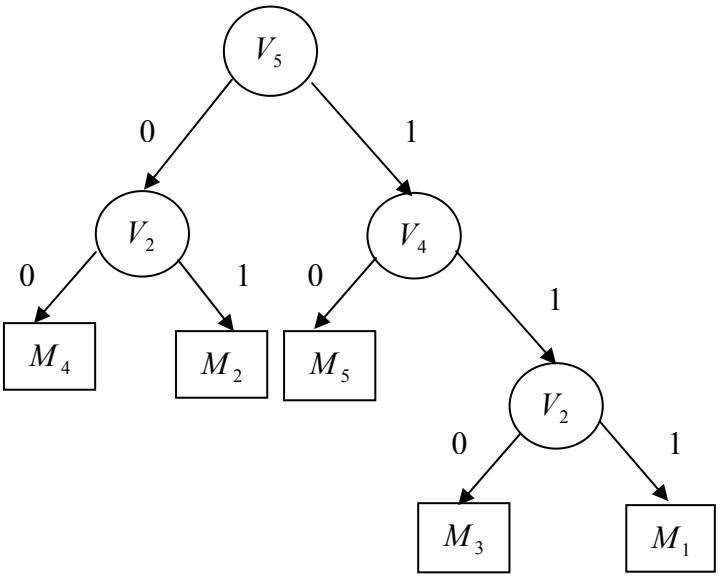

Figure 2. Locating tree of defect modules

2°) The determination of average duration for locating/testing of the modules

To determine T_i time we take in account the probability of execution of each function and the results testing made in according with the specifications for that function.

That is:

$$T_i = \sum_{j=1}^{nr\_f\_\mathrm{mod}_i} (durata\_f_j + test\_rap\_sp\_f_j) * p\_ex\_f_j, \qquad (6)$$

where:

$nr\_f\_\mathrm{mod}_i$ = number of functions from module i tested;

$p\_ex\_f_j =$ probability of execution of function j;

$durata\_f_j =$ execution time of function j;

$test\_rap\_sp\_f_j =$ testing time of function j results in according with the specifications;

We established the next values T.U.(Unites of Time) for used functions(table 3):

Table 3. Execution and testing times in according with the specifications and the times for locating/recovery for the functions used in modules

Function Name	**Execution Time**	Testing time in according with the specifications	**Average locating/recovery time**
Push(z)	30 T.U.	20 T.U.	300 T.U.
Pop( )	25 T.U..	20 T.U.	250 T.U.
Top (z)	20 T.U..	15 T.U.	100 T.U.

Starting from formulas:

$$T_i = \sum_{j=1}^{nr\_f\_\mathrm{mod}_i} (durata\_f_j + test\_rap\_sp\_f_j) * p\_ex\_f_j, \quad \text{(see 6)}$$

$$\theta_i = \sum_{j=1}^{nr\_f\_\mathrm{mod}_i} durata\_loc\_rest\_f_j \quad \text{(7)},$$

where:

$durata\_loc\_rest\_f_j =$ locating/recovery time of function j in module i,

we'll have:

$T_1 =$ execution time for 3 source instructions(6 T. U.) + testing time in according with the specifications (0 T. U.) = 6 T. U.);

$\theta_1 =$ 1 T. U. (a simple instruction if.... then ....else);

$T_2 =$ (30 T. U. + 20 T. U.) *1 = 50 T. U.;

$\theta_2 = 300$ T. U.;

$T_3 =$ (7 T. U.+ 0 T. U.) = 7 T. U.;

$\theta_3 =$ 1 T. U. (a simple instruction if ...then...else);

$T_4 =$ (30+20) · 0,1+(25+20)· 0,8 +(20+15)· 0,1=50·0.1+45·0.8+35·0,1 = 44.5 T. U.;

$\theta_4 =$ 300 · 0,1+250·0,8+100·0,1 = 240 T. U.;

$T_5 =$ (30+20)· 0,8+(25+20)·0,1+(20+15)· 0,1 = 50·0,8+45·0,1+35·0,1 = 48 T. U.;

$\theta_5 =$ 300·0,8+250·0,1+250·0,1 = 290 T. U.

For the locating tree of defect modules from fig. 2, we'll have the next repartition law for the discreet random variable T_{loc} :

$$T_{loc} = \begin{vmatrix} T_2+T_4 & T_2+T_5 & T_2+T_4 & T_2+T_5 & T_4+T_5 \\ +T_5+\theta_1 & +\theta_2 & +T_5+\theta_3 & +\theta_4 & +\theta_5 \\ p_1 & p_2 & p_3 & p_4 & p_5 \end{vmatrix}$$

$$T_{loc\_med} = (T_2 + T_4 + T_5 + \theta_1) \cdot p_1 + (T_2 + T_5 + \theta_2) \cdot p_2 + (T_2 + T_4 + T_5 + \theta_3) \cdot p_3$$
$$+ (T_2 + T_5 + \theta_4) \cdot p_4 + (T_4 + T_5 + \theta_5) \cdot p_5 = (50 + 44.5 + 48 + 1) \cdot 0.193 + (50 + 48 + 300)$$
$$\cdot 0.196 + (50 + 44.5 + 48 + 1) \cdot 0.201 + (50 + 48 + 240) \cdot 0{,}205 + (44{,}5 + 48 + 290) \cdot 0.205 =$$
$$282{,}2495 \quad T.U.$$

3°) The determination of average testing and locating/recovery duration of the modules, using the residual number of defects into a program, with formula (5):

$$T_{loc\_gen\_med} = E \cdot T_{loc\_med} = (\omega_1 + \omega_2 + \omega_3 + \omega_4 + \omega_5) \cdot T_{loc\_med}$$
$$= (0.092 + 0{,}461 + 0{,}123 + 0{,}923 + 0.923) * 282{,}2495$$
$$= 2{,}522 * 282{,}2495 \approx 711{,}833 \text{ T.U.}$$

4°) The determination of the weights and theirs importance (formula 4):

$$\Pi_1 = Tloc_1 \cdot p_1 / T_{loc\_med} = (T_2 + T_4 + T_5 + \theta_1) \cdot p_1 / T_{loc\_med} = 27{,}6955 / 282{,}2495 = 0{,}098;$$
$$\Pi_2 = Tloc_2 \cdot p_2 / T_{loc\_med} = (T_2 + T_5 + \theta_2) \cdot p_2 / T_{loc\_med} = 78{,}008 / 282{,}2495 = 0{,}276;$$
$$\Pi_3 = Tloc_3 \cdot p_2 / T_{loc\_med} = (T_2 + T_4 + T_5 + \theta_3) \cdot p_3 / T_{loc\_med} = 28{,}8435 / 282{,}2495 = 0{,}102;$$
$$\Pi_4 = Tloc_4 \cdot p_4 / T_{loc\_med} = (T_2 + T_5 + \theta_4) \cdot p_4 / T_{loc\_med} = 69.29 / 282{,}2495 = 0{,}245;$$
$$\Pi_5 = Tloc_5 \cdot p_5 / T_{loc\_med} = (T_4 + T_5 + \theta_5) \cdot p_5 / T_{loc\_med} = 78{,}4125 / 282{,}2495 = 0{,}278.$$

The decrease row of Π_k values is:

$\Pi_5, \Pi_2, \Pi_4, \Pi_3, \Pi_1$, and this means that the module with the most locating/testing time is M5 and the module with the least locating/testing time is M1, this thing allowing to designers and programmers to redesign and grow the performances of critical modules(regarding of testing/recovery times).

3. General conclusions

The metrics proposed in this paper give a methodological framework in the field of the functional testing for the software programs.

These metrics have the next capabilities:

- give a good understanding for functional testing and locate/recovery mechanism of the software modules of a program;
- give a better appraisal to the performances of the testing tools and to the skill of the debugging personal(by locating/recovery time of a function into a module);
- identify the modules that need a big locating/testing time (by decrease sorting of the weights), asking a possible redesign for the intensive resources modules.

Bibliography

1. Boehm, J., Saib, B. **Error Seeding Technique Specification,** Prepared under Contract Number NAS 2-10550 by Hughes Aircraft Company, Fullerton and General Research Corporation, Santa Barbara, California, USA, 1980
2. Ghita, A., Ionescu, V. **Metode de calcul in fiabilitate,** Course, Technical Military Academy, Bucharest, 1996
3. Goron, S., **Fiabilitatea Produselor Program,** Universitatea Babes-Bolyai, Cluj-Napoca, Ed. Risoprint, 2000
4. Paulk, M., Curtis, B., s.o., **Capability Maturity Model for Software, Version 1.1,** Software Engineering Institute, Carnegie Mellon University, Pittsburg, Pensylvania, USA, 1993
5. Popescu, M., **Managementul Fiabilitatii Aplicatiilor Software Militare,** PhD Dissertation, Technical Military Academy, Bucharest, 2002
6. **System and Software Reliability Assurance Notebook,** Produced for Rome Laboratory, New York, 1997

[1] Codifications of references:

[BOEH81]	Boehm, J., Saib, B. **Error Seeding Technique Specification,** Prepared under Contract Number NAS 2-10550 by Hughes Aircraft Company, Fullerton and General Research Corporation, Santa Barbara, California, USA, 1980
[GHIT96]	Ghita, A., Ionescu, V. **Metode de calcul in fiabilitate,** Course, Technical Military Academy, Bucharest, 1996
[GORO00]	Goron, S., **Fiabilitatea Produselor Program,** Universitatea Babes-Bolyai, Cluj-Napoca, Ed. Risoprint, 2000
[PAUL93]	Paulk, M., Curtis, B., s.o., **Capability Maturity Model for Software, Version 1.1,** Software Engineering Institute, Carnegie Mellon University, Pittsburg, Pensylvania, USA, 1993
[POPE02]	Popescu, M., **Managementul Fiabilitatii Aplicatiilor Software Militare,** PhD Dissertation, Technical Military Academy, Bucharest, 2002
[ROME97]	**System and Software Reliability Assurance Notebook,** Produced for Rome Laboratory, New York, 1997