Integration of the Functional Testing with the General Theory of the Technical Diagnosis

Mihai POPESCU
Titu Maiorescu University, Bucharest, Romania
mihai_popes2000@yahoo.com

The paper goal is to integrate the software reliability into the general theory of the reliability, this thing being a natural approach, perfectly justified by the core concepts governing the two areas.
Keywords: *Reliability, Faults, Errors, Functional Testing, Technical Diagnosis, Entropy*

1 Introduction

According to [1] reliability means "the property of the product to perform certain functions a given time corresponding to a particular regime and certain operational conditions, technical maintenance, repair, storage and transport", and its operational status is "the state of product in which it is capable of performing all tasks, the functional parameters of the product, called basic parameters, remaining within the confines of the technical documentation".
According to the same references, the event consisting of loss of the functional capability is called failure, and the exit of the basic parameters from limits means failure criterion.
We can see, therefore, the similarity between the general concept of reliability and the reliability of software; we also remark that the software is different from other products because it's criterion of failure is identified in violation by the program of the specifications.
This article aims to adapt the general theory of technical diagnosis, in the manner done by [2], encompassing experience in the field of scientific research and education in the Military Technical Academy, where addressing technical diagnosis was made in an engineering style which eventually lead to development of effective design methodologies. Applying this methodology is subject only to specific information about the structure and reliability of software program and the set of tests available for the diagnosis of this program.
We also note that the calculations required for technical diagnosis were included as specific programs in FIAB; this product was designed and made in the Military Technical Academy in scientific research and educational purposes [2].
A software process can be divided (split) on modules, which in turn can be composed of functions, making them subject to software functional testing against functional specifications that have been formulated with a selected operational profile.
Functional testing of software can be based on general principles of technical diagnosis, because is based on information relating to product structure and appropriate test procedures.
Therefore the results of the general theory of technical diagnosis can be reinterpreted, in the usual sense of functional testing of the software.
The benefits of this approach are:

- Provide a basis for the testing of the properties of detection and locating associated with a set of tests and minimize its;
- Give methodological principles for the tests organization into effective testing program;
- Provide a new basis for the calculating of the matrices of complexity and reliability of software.

Technical diagnosis provides an overall methods, processes and techniques aimed at highlighting the errors occurring in the operation of hardware.
Two technical problems are derived from this general objective of the technical diagnosis: fault detection and locating of errors detected occurred.
Software failure may occur as a result of the

existence of human error in one or more modules, having to do so with single fault or multiple faults (which is the general rule).
The detection of the error that caused the fault is the process of the obtaining of the information about a program failure.
This process is naturally followed by the location of detected errors- this is a process of identifying the module (and function) of the program that is fault.

2 Errors Detection

Definition 1. *Detecting errors in a program is the process by which information is obtained about the existence or absence of errors in the program***.** We note by X a finite set, with Card X = L = 2N, where N is the number of modules of the program, set of possible states of the diagnosed program. It follows therefore:

$X = \{ x_i ;\ i= 0,\ldots,L \}.$ (1)

Definition 2. *A Xi program state is a synthetic information about the state of the program modules.*
About program modules we admit that they can be only one of two states: operational status or condition in which they are suspected to be broken.
A state x_i can then be represented as a vector:

$x_i = (x_{i1}\ x_{i2}\ x_{i3}\ldots x_{iN})'$,

$$x_{ik} = \begin{cases} 1, \text{if the module k is suspected to have errors} \\ 0, \text{if not} \end{cases} \quad (2)$$

It follows therefore that x_0=(0,0,0,...,0) is the state in which all modules are running, x_i=(0,0,0,...,0,1,0,...,0), i=1,...,N are states where the module i is suspected to have errors , and $x_L = (1,1,1,\ldots,1)$ is the state where all modules are suspected to have errors.
For a state x_i from set X we note $I(x_i)$ the set of the modules suspected to have errors in that state; hereby we have the formula:

$I(x_i) = \{k \in \{0,1,\ldots, N\} \mid x_{ik} = 1\}$. (3)

The number of modules suspected of errors in the state x_i will be $n(x_i)$, where:

$n(x_i)$= Card I.(x_i) (4)

We say that two states x_i şi x_j are equivalent ($x_i \sim x_j$) if:

$n(x_i) = n(x_j)$. (5)

We obtain a partitioning of the set X in equivalence classes X_k:

$X = \bigcup_{k=0}^{N} X_k$, (6)

where:

$X_k = \{x_i \mid n(x_i) = k\}$. (7)

To solve the problem of detecting and locating errors in a program we run a series of different tests.

Definition 3. *A test is an experimental process by which an error is detected in a subset of program modules.*
In the context of functional testing, a test is identified by testing the correspondence with the specifications for a specific subprogram of the tested program.
We note by V the set of tests (checks) available for diagnosis:

$V = \{v_j \mid j = 1,\ldots, M\}$, (8)

where M is the number of available tests; M is finite.
The essential information for a test are the modules tested. We represent a test as a vector

$v_j = (v_{j1}, v_{j2}, v_{j3},\ldots, v_{jN})$, (9)

where:

$$v_{jk} = \begin{cases} 1, \text{if the module k is tested by test } v_j; \\ 0, \text{if not.} \end{cases} \quad (10)$$

The set of tested modules we note $J(v_j)$ and we have:

$J(v_j) = \{k \in \{1,2,\ldots, N\} \mid v_{jk} = 1\}$. (11)

In choosing a set of tests it must be ensured that for any program module it should exist a

test for that module.

Proposition 1. *Necessary and sufficient condition for a set of tests V to ensure detection of any error is that for any program module to be at least one test to test that module.*

The following sentence is equivalent with the sentence 1**.**

Proposition 2. *For any k=1,...,N it exists a test* $v_j \in V$ *so* $v_{jk} = 1$.

Definition 4. *The matrix* $V = (v_{jk})$, *j=1,...,M , k=1,...,N is called test matrix.*

The introducing of the concept of a matrix of tests is advantageous, in terms of calculus, for studying the properties of the diagnostic process.

In these circumstances, propositions 1 and 2 are equivalent to the following sentence:

Proposition 3. *Necessary and sufficient condition for a set of tests V to ensure detection of any error is the test matrix V has no null columns.*

If a set of tests V ensures detection of any error, we can find then a included matrix of the tests matrix V so the propositions 1 and 3 are met with a minimum number of tests.

The included matrix of V is noted V^* and is called reduced matrix.

Further it is recognized that it operates only in the form of reduced matrix V.

3 Locating Errors

In terms of evolutionary time, the state program can be characterized as a random variableξ_t, t∈T with values in the set of states X.

If the diagnostic problems are solved in succession, we can use as time set the set of natural numbers (T=N). It follows that (ξ_t), t ∈T, is a random process that can be treated as a Markov process with finite set of states.

Definition 5. *The locating of the errors is the experimental process that identifies fault software modules*.

Locating process can be executed simultaneously with or after error detection procedure if there faulting modules exist in program.

In this case we can write:

$\xi_t \in X \setminus X_0$.

If there are only single errors and tracking activities are executed sequentially in time, then usually $\xi_0 \in X_L$ (leaving the original state in which all program modules are suspected to have errors) and it aims to reach in $\xi_t \in X_1$ state, which means locating faulty module.

The software has, usually, several modules with errors and more errors for each module.

The hardware is operating in practice with individual breakdowns, even when there are multiple faults, because in this case there are not theoretical results, implemented, similar to those of single faults.

Therefore, to eliminate multiple faults successively apply the methodology of single faults until the detecting no signals the presence of faults [3].

By analogy to locate multiple errors in the software we will proceed in the same manner developing the theory of individual errors and implementing, repeatedly, the procedures established for locating, until all errors are eliminated.

This procedure can be justified practically by the fact that the multiple errors (faults) are not manifested simultaneous necessarily so for every run of a program they can show as individual errors

Working assumptions will be described below.

Assumption 1. *We consider only single errors.*

Testing process starts from initial state $\xi_0 \in X_L$ and continues until $\xi_t \in X \setminus (X_0 \bigcup X_1)$, when we locate the module containing errors.

Each state $x_i \in X$ has a set of suspected modules I(x_i). For each test $v_i \in V$ we have a set of tested modules $J(v_j)$. After the test $v_i \in V$ the set of suspected modules is:

a) $I(x_i) \bigcap J(v_j)$ if the error exists;

b) $I(x_i) \setminus J(v_j)$ if the error not exists.

Assumption 2. *The tests* $v_j \in V$ *certainly give the modules containing errors.*

For a test v_j to ensure the restriction of suspicious modules it must include at least a suspicious modules and no simultaneously all suspicious modules. In other words, a test v_j

$\in$ V must meet the following condition:

$\varnothing \subset I(x_i) \bigcap J(v_j) \subset I(x_i)$.

We note with $V^*(x_i)$ the set of tests $v_j \in V$ that have the property stated above and which defines clearly the relationship:

$$V^*(x_i) = \{v_j \in V \mid \varnothing \subset I(x_i) \bigcap J(v_j) \subset I(x_i)\}. \qquad (12)$$

Definition 6. *The set $V^*(x_i)$ is called the set of the useful tests in the state x_i.*

Its importance is apparent from the following sentence:

Proposition 4. *A set of tests V provides the location of any errors if*

$V^*(x_i) \neq \varnothing$, $(\forall) x_i \in X \setminus (X_0 \bigcup X_1)$.

In other words a set of tests provides the location of any errors if tests are useful whenever there is more than one suspect module.

The following sentence is equivalent to proposition 4.

Proposition 5. *A set of tests V provides the location of any errors if*

$V^*(x_i) \neq \varnothing, (\forall) x_i \in X_2$.

In other words, a set of tests provides the location of any errors if any pair of suspicious modules can be detected.

This property is stated in [2] : A set of tests provides the location of any fault detected if two faults are two distinctive.

The following sentence expresses the matrix form of the sentence 5.

Proposition 6. *Necessary and sufficient condition for a set of tests V to ensure the location of any detected errors is that the test matrix V has no identical columns.*

From matrix V we can choose a included matrix V^* ensuring proposition 6 with a minimum number of lines. In this way we can ensure the location of any errors detected by a minimum number of checks.

The included matrix V^* with this property is called also reduced matrix, and further we assume that the reduced form of V is used.

4 Automatic Signalling of the States

In terms of organization of the testing process for errors detected in the program we can distinguish two different cases:

a. The tests run simultaneously. In this situation we can signal the states of program that may indicate at any time if the program is without error or an error is detected in the program.

b. Tests are executed sequentially (successively). In this situation the locating process can be executed sequentially to locate the error detected. Automatic signalling of the states is possible where the program is continually monitored by simultaneous tests in order to indicate any time if no errors and if so, what is the perfect module. Its implementation is done when the assumptions 1 and 2 are satisfied:

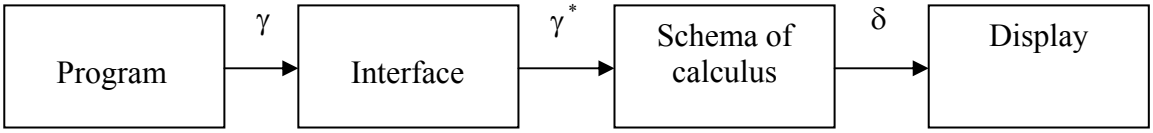

Fig. 1. Scheme of automate signalling of the states of a program (hardware variant)

In this case, a test of V is equivalent to measuring a functional parameter γ_j of the system and comparing it with a certified field Γ_{0_j}, j=1,...,M. To use a such information the testing scheme should be structured as fig.1, where $\gamma = (\gamma_1, \gamma_2, ..., \gamma_M)$ is the vector for the compared parameters and $\gamma^* = (\gamma^*_1, \gamma^*_2, ..., \gamma^*_M)$ is the vector with the results of this comparison, where: :

$$\gamma^*_j = \begin{cases} 1, \text{if } \gamma_j \notin \Gamma^o{}_j \\ 0, \text{if } \gamma_j \in \Gamma^0{}_j \end{cases} \quad \text{..... (13)}$$

and the calculus scheme realises the correspondence between γ^* and δ signals; δ has the structure: $\delta=(\delta_0,\delta_1,\delta_2,...,\delta_N)$, where:

$$\delta_0 = \begin{cases} 1, \text{if the program has no errors (is in state } x_0) \\ 0, \text{if the program is in any state } x_i, i=1,...,N) \end{cases} \quad (14)$$

and

$$\delta_i = \begin{cases} 1, \text{if the module i is fault} \\ 0, \text{the module i is not fault} \end{cases} \quad (15)$$

for i=1,...,N.

Table 1. The logical function of the calculus scheme

Parameters	γ^*			δ			
States	γ^*_1	...	γ^*_M	δ_0	δ_1		δ_N
0	0	...	0	Unit matrix			
1	Transposed matrix V						
...							
N							

The hardware implementation of the scheme of calculation is done using a combinational circuit whose function is to distinguish the fault module(s).

Given that the matrix V provides the detection and locating of any errors, the transposed matrix V will not have the identical null or identical lines.

Logic equations for δ_0, δ_1,…, δ_N values are obtained from Karnaugh diagrams and their minimize (Table 1).

FIAB product contains a section on automatic signalling of the status [2], providing at output the both reduced matrix V' and table 1 for a tests matrix V.

Figure 2 proposes a software implementation of the scheme for automatic signalling of the program states (fig.1), where the following notations are used:

- NL = number of rows of matrix V' ;
- NC = number of columns of matrix V' ;
- Y* = the vector that keeps, in ascending order, the number of lines remaining in reduced matrix V 'of the original matrix V;
- V't= transposed matrix of the matrix V'.

5 Locating Errors in Successive Tests

The notion most representative for this case is the concept of locating program.

Definition 7. *Locating program is a sequence of tests applied to a program in order to locate the error detected.*

There are two basic classes of software locating

- **fixed programs**, when the tests succeed in a fixed sequence until the error detected is located;
- **adaptive programs**, when the next test is determinate by the results of the previous tests.

Effectiveness of a program that locates the error detected can be characterized by many indicators, from which:

- **the average number of steps** to locate error;
- **the average time** for location;
- **the average cost** of locating process ;
- **the average amount of information** brought by the tests.

In what it follows we will present the main concepts related to adaptive locating programs that use as a criterion of efficiency the average amount of information brought by the tests, the key advantage that provides the most convenient implementation

Whatever is the criterion of efficiency achieved by the locating program, it is necessary to determine a priori probability of failure of software modules.

If we note with p_i a priori probability of failure of the module i from system, with D_i the event "module i is fault", and with D the event "the program is fault, then we have the relations:

$$D = \bigcup_{i=1}^{N} D_i \quad (16)$$

and $p_i = P(D_i / D),$ (17)

resulting

$$p_i = P(D_i) / P(D). \quad (18)$$

Probabilities p_i can be determined either experimentally or analytical.

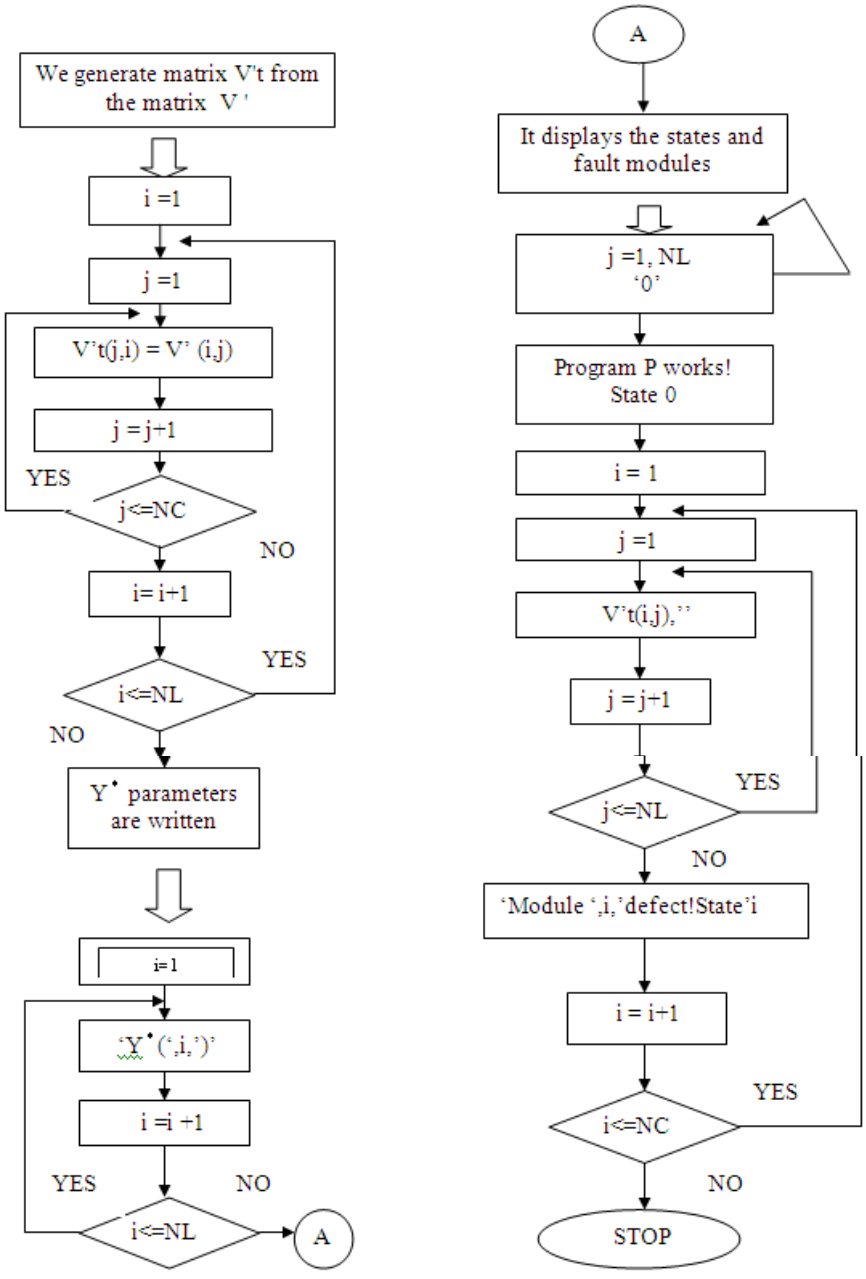

Fig. 2. Scheme of automate signalling of the states of a program (software variant)

a) Experimentally determining of the p_i values

After we analyze N cases when the program was fault, we determine the number of cases M_i when the module i was fault. We can say, under the law of large numbers, that M_i/N converges in probability to p_i and therefore we can approximate p_i even with this fraction

$$p_i \cong M_i / N. \quad (19)$$

Obviously the following properties are satisfied (assuming single fault):

$$1)\ p_i \geq 0, i = 1,...,N; \quad (20)$$

$$2)\sum_{i=1}^{N} p_i = 1. \quad (21)$$

b) Analytical determining of the p_i values

Assuming that the failure of the program modules follows a exponential distribution law of life time, i.e. it complies with the model Jelinski-Moranda (it supposes that are N1 errors in program at the beginning, each independent of each other and equally likely to cause a failure during testing; the number N1 of error is estimated by maximum likelihood method; the errors are not removed until a fatal error appears, when the accumulated errors group is extirpated and no new ones are introduced during debugging), we have:

$$P(D_i) = 1 - e^{-\lambda_i t} \cong \lambda_i t \quad (22)$$

if the condition $\lambda t << 1$ is realized , and for the entire program:

$$P(D) = 1 - e^{-\lambda t} \cong \lambda t, \quad (23)$$

where λ_i is the intensity of failure of the module i, and λ is the intensity of failure of the program:

$$\lambda = \sum_{i=1}^{N} \lambda_i \quad (24)$$

For the mission duration t, so that $\lambda_i t << 1$ and $\lambda t << 1$, we can approximate by virtue of relations (22) and (23) a priori probability with the relationship:

$$p_i \cong \lambda_i / \lambda \quad (25)$$

Values p_i thus obtained also satisfy conditions 1) and 2) of case a).

Failure intensities calculation of the modules will be made to the formula proposed in [3]:

$$\lambda_k = r * Me * \omega_k / I_k \quad (26)$$

where:

$Me=4{,}2*10^{-7}$ is the Musa exposure rate of faults;

r = r processor speed, in instructions / s - can be determined from benchmarking programs or technical characteristics of the computer offered for sale;

ω_k = number of faults contained by module k. He is determined transforming source instructions written in a programming language into functional points and then determining the number of faults as CMM Level (Capability Maturity Model) selected [4];

I_k = number of lines of executable code k * expansion rate[4].

Random nature of a locating program is given by the results of the tests.

Definition 8. *The result of a test is a information that confirms/not confirms the susceptibility about the tested modules.*

The result of the testing we note $r_{ij}(k)$ and it represents the result of test vj in the state x_i , when the module k is fault. Therefore it is a binary random variable, defined by:

$$r_{ij}(k) = \begin{cases} 1, \text{if } k \in I(x_i) \cap J(v_j) \text{ (it confirmes the susceptibility)} \\ 0, \text{if } k \in I(x_i) \setminus J(v_j) \text{ (it not confirmes the susceptibility)} \end{cases} \quad (27)$$

So r_{ij}(k) is a discreet random variable having the next repartition table:

$$r_{ij}(.) : \begin{pmatrix} 0 & 1 \\ Q_{ij} & P_{ij} \end{pmatrix}, \quad (28)$$

where:

$$P_{ij} = P(r_{ij} = 1), \quad (29)$$

$$Q_{ij} = P(r_{ij} = 0), \quad (30)$$

which we'll have the relations:

$$P_{ij} = \frac{\sum_{k \in I(x_i) \cap J(v_j)} p_k}{\sum_{k \in I(x_i)} p_k}, \quad (31)$$

$$Q_{ij} = \frac{\sum_{k \in I(x_i) \setminus J(v_j)} p_k}{\sum_{k \in I(x_i)} p_k}. \quad (32)$$

We'll have, obviously:

$$Q_{ij} = 1 - P_{ij}. \quad (33)$$

We'll note x_i^1 the information obtained from x_i state, when the susceptibility is confirmed (respectively $r_{ij} = 1$) and with x_i^0 the information obtained from x_i state , when the susceptibility is not confirmed (respective $r_{ij} = 0$).

We note $H(x_i)$ the entropy of the state x_i, and $H(x_i^1)$, respectively $H(x_i^0)$, the entropy of the state x_i after the test v_j, with confirmation/not confirmation of the susceptibility.

We note $p_k(x_i)$ the fault probability of module k, when it belongs to the set $I(x_i)$ and we'll have:

$$p_k(x_i) = \frac{p_k}{\sum_{k \in I(x_i)} p_k} \quad (34)$$

In these situations we will have the entropies:

$$H(x_i) = -\sum_{k \in I(x_i)} p_k(x_i) \cdot \log p_k(x_i), \quad (35)$$

$$H(x_i^1) = -\sum_{k \in I(x_i^1)} p_k(x_i^1) \cdot \log p_k(x_i^1), \quad (36)$$

$$H(x_i^0) = -\sum_{k \in I(x_i^1)} p_k(x_i^0) \cdot \log p_k(x_i^0), \quad (37)$$

We'll note χ_{ij} the entropy after testing. χ_{ij} is a discreet random variable with the next repartition table:

$$\chi_{ij} : \begin{pmatrix} H(x_i^0) & H(x_i^1) \\ Q_{ij} & P_{ij} \end{pmatrix}. \quad (38)$$

The average entropy after testing will be the average value of χ_{ij}, noted with H_{ij}:

$$H_{ij} = E[\chi_{ij}] = H(x_i^1)P_{ij} + H(x_i^0)Q_{ij}. \quad (39)$$

The average information brought by test v_j in the state x_i, noted with $I(v_j / x_i)$, will be:

$$I(v_j / x_i) = H(x_i) - H_{ij}. \quad (40)$$

To determine these values we'll first rewrite the values for $p_k(x_i^1)$ and $p_k(x_i^0)$ which, according to (33), are:

$$p_k(x_i^1) = \frac{p_k}{\sum_{k \in I(x_i^1)} p_k} = \frac{p_k}{\sum_{k \in I(x_i^1)} p_k} \cdot \frac{\sum_{k \in I(x_i)} p_k}{\sum_{k \in I(x_i)} p_k} = \frac{p_k(x_i)}{P_{ij}} \quad (41)$$

respectively

$$p_k(x_i^0) = \frac{p_k}{\sum_{k \in I(x_i^0)} p_k} = \frac{p_k}{\sum_{k \in I(x_i^0)} p_k} \cdot \frac{\sum_{k \in I(x_i)} p_k}{\sum_{k \in I(x_i)} p_k} = \frac{p_k(x_i)}{Q_{ij}}, \quad (42)$$

where:

$$I(x_i^1) = I(x_i) \cap J(v_j), \quad (43)$$

$$I(x_i^0) = I(x_i) \setminus J(v_j) \quad (44)$$

and we used relations (32) and (34).

We'll calculate $H(x_i^1)$ and $H(x_i^0)$:

$$H(x_i^1) = -\sum_{k \in I(x_i^1)} p_k(x_i^1) \log p_k(x_i^1) = -\sum_{k \in I(x_i^1)} \frac{p_k(x_i)}{P_{ij}} \cdot \log \left[\frac{p_k(x_i)}{P_{ij}} \right] =$$

$$= -\frac{1}{P_{ij}} \sum_{k \in I(x_i^1)} p_k(x_i) \left[\log p_k(x_i) - \log P_{ij} \right] = (45)$$

$$= -\frac{1}{P_{ij}} \left[\sum_{k \in I(x_i^1)} p_k(x_i) \log p_k(x_i) - P_{ij} \log P_{ij} \right].$$

We'll also have:

$$H(x_i^0) = -\sum_{k\in I(x_i^0)} p_k(x_i^0)\log p_k(x_i^0) = -\sum_{k\in I(x_i^0)} \frac{p_k(x_i)}{Q_{ij}}\cdot\log\left[\frac{p_k(x_i)}{Q_{ij}}\right] =$$
$$= -\frac{1}{Q_{ij}}\sum_{k\in I(x_i^0)} p_k(x_i)\left[\log p_k(x_i) - \log Q_{ij}\right] = \quad (46)$$
$$= -\frac{1}{Q_{ij}}\left[\sum_{k\in I(x_i^0)} p_k(x_i)\log p_k(x_i) - Q_{ij}\log Q_{ij}\right].$$

Replacing in (38) the values above obtained, we'll have:

$$H_{ij} = H(x_i) + P_{ij}\log P_{ij} + Q_{ij}\log Q_{ij}, \quad (47)$$

resulting:

$$I(v_j / x_i) = -\left[P_{ij}\log P_{ij} + Q_{ij}\log Q_{ij}\right]. \quad (48)$$

That is to say, in natural language, that [2]:

Theorem 1. *The average information brought by a test is equal with the entropy of the testing result***.**

Immediate conclusion is that at each step of testing it should be chosen that test which maximizes the average amount of information given by (47).

Maximum value I (ν_j/x_i) is obtained for that test which has the value of P_{ij} the closest of 0.5. Choosing the test in the step i will be subject to the results obtained in previous steps.

The locating program of the detected error in software can be represented by a tree graph Γ=(V,R) whose nodes are the running tests and arcs represent the results.

Origin of the graph is the test running first time, and each peak of the graph represents the fault module of the program.

For a locating program, as mentioned above, the tests may be succeed in a fixed sequence or in an order determined by the test result at each step.

Figure 3 shows the principle of construction of the graph for the locating of the error detected when the locating process is optimized by the criterion of average quantity of information brought by tests.

The locating algorithm is given below:

Step 1. For each test v_j, from the set of helpful tests V*(x_i), is calculated the P_{ij} value then the calculated values will be included in the last column of the table.

Step 2. Determine the index value j * for which the value P_{ij} of the last column is the closest to 0.5 and run the test ν_j* ; after the test execution two situations occur: or not confirm susceptibility (r = 0 or r = 1) .

Step 3. If, after the testing, the fault module is among suspected modules (confirmation) we'll build a new table that will contain on rows the checks that belong to V*(x_i^1) and the original columns will keep only the columns that belong to I(x_i^1) (the modules which were suspected to be fault were tested and suspicion was confirmed).

Step 4. If, after the testing, the fault module is not among suspected modules (not confirmation) we'll build a new table that will contain on rows the checks that belong to V*(x_i^0) and the original columns will keep only the columns that belong to I(x_i^0) (the modules which were suspected to be fault were tested and suspicion was not confirmed).

Continue actions from step 1 to 4 until the fault module is localized. Presented algorithm is implemented in FIAB utility [2].

The locating time of an error detected in a program is a discrete random variable which we note, as [1], with T_{rest}, having the following distribution table:

$$T_{rest}:\begin{pmatrix}\Theta_1 & \Theta_2 \ldots \Theta_N\\ p_1 & p_2 \ldots p_N\end{pmatrix}, \quad (49)$$

where Θ_i is the locating time of a error existing in module i. This value can be calculated using the locating graph for detected error. In this case, in the graph Γ is a single subgraf Γ_i that starts from root to i representing the sequence of tests until the location of the error detected i. We make the following notations:

$$J_i = \{j / v_j \in \Gamma_i\} \quad (50)$$

So J_i is the set of indices of the tests included on the branch Γ_i. If T_j is the time of test j, we will have:

$$\Theta_i = \sum_{j \in J_i} T_j. \qquad (51)$$

In these circumstances, the distribution of the locating time is completely determinate.
We note, according to the standard [1], with Trest.med the average time for the locating of the error detected, respectively $T_{rest.med} = M[T_{rest}]$.
Then:

$$T_{rest.med} = \sum_{i=1}^{N} p_i \cdot \Theta_i \qquad (52)$$

and respectively

$$T_{rest.med} = \sum_{i=1}^{N} p_i \cdot \sum_{j \in J_i} T_j. \qquad (53)$$

Observation 1. *If in the formula (51) T_j is replaced with 1, the value $T_{rest.med}$ thus obtained will represent the average number of steps required for the locating of the error detected.*

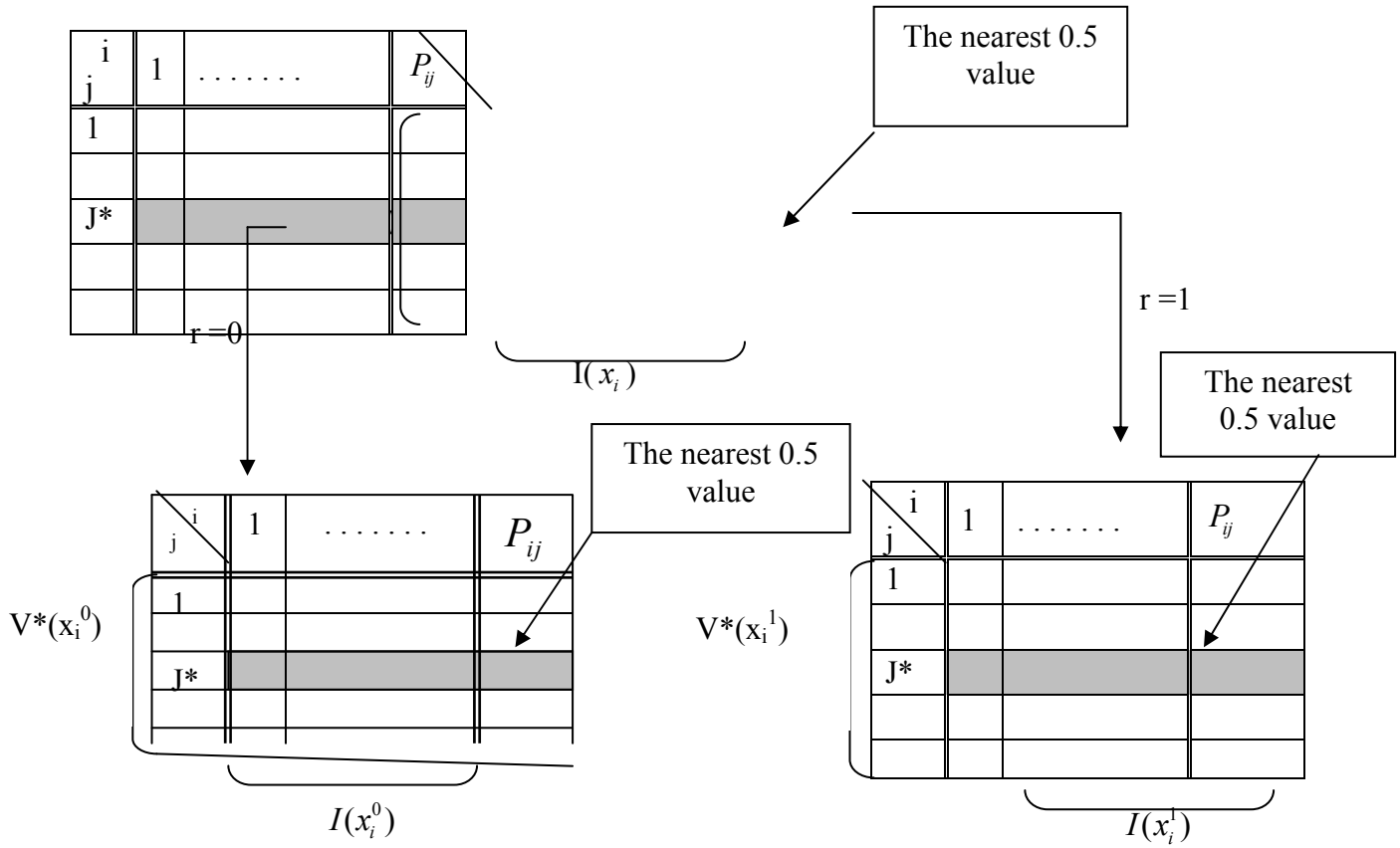

Fig. 3. Building of the graph for the locating of the detected error optimized by the criterion of average quantity of information brought by tests

6 Conclusions

The locating program built contains all methodological information needed to execute the locating process. It must then complete with details of the tests contained in it: what parameters are tested, and which are normal specifications (i.e. specifying the criterion of failure).
We emphasize once again the generality of this methodology, the application of which is subject only to the existence of specific information on program structure, the a priori probability of failure of modules and to the set of tests available for the diagnosis of the program

References

[1] STPM 040066-91, STPM 040070-91,

Standarde Profesionale Militare, Aparatură, instrumente, dispozitive şi echipamente cu destinaţie militară.

[2] A. Ghiţă, V. Ionescu and M. Bică, *Metode de calcul în mentenabilitate*, Editura A.T.M., 2000.

[3] M. Popescu, *Managementul fiabilităţii aplicaţiilor software*, *Teză de doctorat*, A.T.M., Bucureşti, 2002.

[4] *System and Software Reliability Assurance Notebook*, Produced for Rome Laboratory, New York, 2007.

Mihai POPESCU has graduated the Faculty of Electronics and Electrical from Military Technical Academy, Bucharest, in 1983; he holds a PhD diploma in Computer Science from 2002 and he had gone through two didactic positions since 1998 when he joined the staff of the Bucharest Military Technical Academy- lecturer in 1998 and assistant professor in 2002. Currently he is assistant professor of Titu Maiorescu University, Bucharest, at Faculty of Informatics. He is the author of more than 6 books and over 55 journal articles in the field of software reliability, software metrics and data bases.