BIULETYN INSTYTUTU SYSTEMÓW INFORMATYCZNYCH **13** 25–36 (2014)

A multifaceted model for software reliability prediction during testing

R. PEŁKA
radoslaw.pelka@wat.edu.pl

Institute of Computer and Information Systems
Faculty of Cybernetics, Military University of Technology
Kaliskiego Str. 2, 00-908 Warsaw, Poland

Analysis of software reliability plays an important role in quality assurance plan realization during software development. By monitoring changes of evaluated reliability in relation to quality objectives it is possible to analyze current situation in respect to agreed requirements and initiate appropriate actions when needed to secure fulfilling of the goals. The use of software reliability growth models as the only method for reliability evaluation seems to be too much simplified approach. Such approach, based solely on fault detection history, may in some circumstances be risky and lead to significantly wrong decisions related to the software validation process. Taking possible pros and cons into account the model described in this paper is proposed to use a number of additional information concerning the software being tested and the validation process itself, to produce more accurate outcomes from the reliability analysis. The produced outcome gives an appropriate feedback for a decision makers, taking into account assumed software development process characteristic. Integral part of the presented approach is devoted to reliability characteristics of a system being tested in parallel by several independent teams.

Keywords: reliability, software, modeling, testing.

1. Introduction

Software reliability analysis plays an important role in overall quality assurance plan in software development process. Nowadays, when business realities force companies to be more competitive when it comes to faster deliveries of a new software to the market, software reliability remains a crucial factor which determines the final success of a product. Such situation provoke existence of a development process optimization procedures that incorporate reliability objectives as a main criteria. Such approach may be potentially good but on the other hand it may also be dangerous when reliability findings are not proper. To make sure that produced reliability evaluations can be rely on, it is crucial to secure that the applied methods allow to incorporate possibly large range of important aspects related to software verification process and system under testing as such. This kind of approach is more demanding for persons performing quality analysis because it entails a need to collect, prepare and process larger amount of data, compared to situation when only software reliability growth models (SRGMs) are used [1]. Successful application of SRGMs has been proved many times [2]. Therefore, the approach proposed in this paper incorporates use of an SRGM as a method for software reliability prediction, however, taking all the related disadvantages into account [3], [4], additional extensions have been proposed. The extensions are used to make it possible, in a given moment of software validation process, to produce more accurate outcomes for a decisive persons, based on results provided by SRGM but processed in accordance with taken assumptions. The main goal for this version of the model was to verify methods of information synthesis and influence of such approach on the quality and financial results of a software development project execution when software validation is supposed to be conducted under strictly defined reliability objectives.

The subject of interest and research area for the proposed approach are large systems consisting of many functional modules, implementing logic for various and complex tasks. In case of such systems it is a common situation that implementation of functional extension to the software is performed simultaneously by several development teams. To a large extend the teams may work independently of each other, practically up to the final integration phase when effects of their work are joined with system being prepared for a customer. From software reliability analysis point of view each such individual process being realized by a single team is important. As well, important is a main process which consists of all the individual processes.

Taking into account the specificity of software development process conducted in the mentioned way, appropriate methods for an individual process are described in chapters 2–6, and methods for a main process are described in chapter 7. Chapter 8 contains summary of the results of application of the model in a real software development project.

2. Software reliability evaluation based on faults detection history

A starting point in the proposed model is to evaluate software reliability, based on information concerning history of faults detection during system testing. For this purpose the Musa and Okumoto software reliability growth model [5], based on non-homogeneous Poisson process theory, was used. This model was selected due to proven effectiveness in practical application and satisfactory level of fit of the model to data representing history of defects detection in the examined software. In order to evaluate the fit of model to available data, three criteria which are widely used for the purpose of SRGMs comparative analysis [6] can be applied. The criteria are the mean squared errors (1), the predictive-ratio risk (2), and Akaike's information criterion (3).

$$MSE = \frac{\sum_{i=1}^{u}[m(t_i)-y_i]^2}{u-N}, \tag{1}$$

where:
y_i – number of faults detected until time t_i;
t_i – time of i-th fault detection;
u – number of data concerning fault detection times, $u > N$;
N – number of model parameters;
$m(t_i)$ – expected cumulative number of faults at time t_i, $i = \overline{1,u}$.

$$PRR = \sum_{i=1}^{u}\left(\frac{m(t_i)-y_i}{m(t_i)}\right)^2, \tag{2}$$

where:
y_i – number of faults detected until time t_i;
t_i – time of i-th fault detection;
u – number of data concerning fault detection times;
$m(t_i)$ – expected cumulative number of faults at time t_i, $i = \overline{1,u}$.

$$AIC = 2q - 2\ln(L), \tag{3}$$

where:
q – number of model parameters;
L – maximum likelihood function of the selected model.

The Musa-Okumoto model belongs to the class of models with infinite number of faults. Due to the form of the mean value function (4) this model is classified as logarithmic model. In such case intensity of failures decreases exponentially, along with detection of subsequent faults. Therefore, tendency to detect more faults in the early phase of testing is incorporated in the model.

$$m(t) = \beta_0 \ln(\beta_1 t + 1), \tag{4}$$

where:

$$\beta_0 = \frac{1}{\theta}$$

$$\beta_1 = \lambda_0\theta$$

λ_0 – initial failure intensity;
θ – rate of reduction in the normalized failure intensity per failure, $\theta > 0$.

For the Musa-Okumoto model appropriate form of the maximum likelihood function (5), required for Akaike's criterion calculation, was determined based on general form of Poisson distribution probability density function.

$$L(\beta_0,\beta_1|t_1,t_2,\dots,t_u) = u\ln\beta_0 + u\ln\beta_1 - \dots\dots$$
$$-\beta_0\ln(\beta_1 t_u + 1) - \sum_{i=1}^{u}\ln(\beta_1 t_i + 1), \tag{5}$$

where:
t_i – time of i-th fault detection;
t_n – time of u-th fault detection;
u – number of data concerning fault detection times.

For a given moment in software validation process appropriate values for β_0 and β_1 parameters can be determined by estimators based on maximum likelihood method. Estimators obtained by this method are usually characterized by at least consistency, asymptotic normality and asymptotic efficiency. Having the model parameters evaluated it is possible to determine value of conditional reliability function (6) for time period $t + x$.

$$R(x|t) = \frac{R(t+x)}{R(t)} = \left[\frac{\beta_1 t+1}{\beta_1(t+x)+1}\right]^{\beta_o}, \tag{6}$$

where:
$R(t) = e^{-m(t)}$,
$m(t)$ is defined by (4).

3. Reliability evaluation risk factor

When software reliability evaluation for a given time horizon is determined, it is then a relevant question how much the evaluation is credible in context of the system under testing as such and current stage of the validation process. To be able to incorporate aspects that have the potential to influence results of the ongoing reliability analysis, a reliability evaluation risk factor is proposed. The risk factor is supposed to be built on information concerning risk of reliability evaluations from a single module perspective, together with information about significance of the module from system perspective (a module weight). The risk factor is supposed to be a function of time, where time is discretized, with step equal k. A step length is a decision variable and can be set to e.g. an hour, a day or a week. A step length shall be set the way that its value corresponds to the characteristic and pace of the validation process realization. It should be relatively shorter than whole planned validation period and relatively longer than execution time of a single test case.

Required information about risk from a single module perspective is built on data concerning test coverage and adequacy of the number of faults detected in a module compare to the expected value. The expected number of faults for a given module is determined based on historical data analysis, taking into account scale of current development project (7). It is assumed that continues development of software from a system module, by using the same programming paradigm in each of the development projects, gives enough argument to perceive the software as to be homogeneous from reliability perspective. In case of lack of information concerning faults detected in a given module, the expected number of faults can be determined by applying method based on a program volume, proposed by Halstead [7].

$$o_i = \left\lceil L_i \cdot \frac{\sum_{j=1}^{m} y_{ij}}{\sum_{j=1}^{m} Y_{ij}} \right\rceil , \tag{7}$$

where:
m – number of historical projects, $m \epsilon N$;
y_{ij} – number of faults in i-th module from j-th historical project, $y_{ij} \in N \cup \{0\}$, $i = \overline{1,n}$, $j = \overline{1,m}$;
L_i – number of new or modified lines of code in i-th module from current project, $L_i \epsilon N$, $i = \overline{1,n}$;
Y_{ij} – number of new or modified lines of code in i-th module from j-th historical project, $Y_{ij} \epsilon N$, $i = \overline{1,n}$, $j = \overline{1,m}$.

Equation (8) shows formula of the function used to assess value of adequacy of the number of faults detected in a given module, in a given moment of the validation process (k), compare to the expected value produced by formula (7). The constant a is a decisive variable whose value shall express the belief of a decisive person about importance of such a fact that number of faults detected in a module differ from the expected value. Value of constant a shall basically not exceed value 2. Higher values lead to situation when even small deviation from the expected value causes significant increase of the risk factor value.

$$A_i(k) = \begin{cases} a^{d_i(k)}, \ when \ d_i(k) < 0 \\ a^{-d_i(k)}, \ when \ d_i(k) \geq 0 \end{cases} , \tag{8}$$

where:

$$d_i(k) = o_i - p_i^{(k)},$$

o_i – expected number of faults for the i-th module, $i = \overline{1,n}$;
$p_i^{(k)}$ – number of faults detected in the i-th module until end of k step, $i = \overline{1,n}$;
a – a constant influencing the shape of adequacy function, $a \geq 1$.

Equation (9) shows formula of the risk factor for a single i-th module modified in current development project. The formula was constructed the way that it takes into account deviation of the number of faults detected in a given module during software validation, from the number of faults expected for this module. It was assumed that as long as there are still some planned but not executed test cases that covers functionality provided by a given module (note that 0 means 0% coverage while 1 means 100% coverage), value of the risk factor for this module is always higher than zero. When number of faults detected in a given module differ from the expected value determined based on historical data, the risk factor value is additionally increased, proportionally to the value of the adequacy factor. For modules not modified in current development project, the risk factor is assumed to be equal zero.

$$h_i(k) = (1 - c_i^{(k)})^{A_i(k)} , \tag{9}$$

where:
$c_i^{(k)}$ – test coverage of the i-th module until end of k step, $c_i^{(k)} \in [0,1]$, $i = \overline{1,n}$;
n – number of modules in the system, $n \in N$.

To be able to determine necessary values of weights of the modules, all the modules shall be classified based on code complexity and functional criticality analysis. This way, appropriate weight value can be given for particular class of modules. Equation (10) is used to determine a numerical value reflecting importance of i-th module from the system reliability examination point of view.

$$I_i = Z_i(1{,}25 - 0{,}25F_i)\ , \qquad (10)$$

where:
Z_i – complexity of i-th module, $Z_i \in [0,1]$, $i = \overline{1,n}$;
F_i – functional criticality of i-th module, $F_i \in \{1,2,3,4\}$, $i = \overline{1,n}$;
n – number of modules in the system, $n \in N$.

Appropriate values of functional criticality (F_i) for all modules are determined by experts having extensive knowledge about examined system. The main criterion here is the impact of a failure in a given module on overall ability of the system to perform its tasks. The meaning of particular values used for expression of the functional criticality is as follows:
1 – high importance module;
2 – normal importance module;
3 – low importance module;
4 – auxiliary module.

Complexity of a given module (Z_i) is determined based on a combination of selected code complexity metrics. The selected metrics shall be appropriate for the type of examined code and shall be characterized by low level of mutual correlation, to maximize effectiveness of their use in the decisive process. In this research the McCabe's cyclomatic complexity [8] and data flow complexity metrics were used. The latter metric is represented by equation (11). It is inspired by data flow complexity concept presented by Henry and Kafura [9].

$$z_i(PD) = (lid_i + liz_i) \cdot (ldd_i + ldz_i + ldp_i), \qquad (11)$$

where:
lid_i – number of interfaces incoming to module, $lid_i \in N \cup \{0\}$, $i = \overline{1,n}$;
liz_i – number of interfaces outgoing from module, $liz_i \in N \cup \{0\}$, $i = \overline{1,n}$;
ldd_i – number of data incoming to module, $ldd_i \in N \cup \{0\}$, $i = \overline{1,n}$;
ldz_i – number of data outgoing from module, $ldz_i \in N \cup \{0\}$, $i = \overline{1,n}$;
ldp_i – number of permanent data maintained by module, $ldp_i \in N \cup \{0\}$, $i = \overline{1,n}$;
n – number of modules in the system, $n \in N$.

Each of the used metrics focuses on different aspects of software engineering, thus they characterize complexity of a given module in a different manner. Due to that it is possible to achieve relatively better evaluation of the overall module complexity, compare to situation when used metrics belong to the same class. The overall complexity for a given module is defined as a product of values given by single metrics. Having the overall complexity calculated for all the system modules, all the values are then normalized to range [0, 1]. The results of normalization are then used as module complexity in equation (10).

When values of the importance factor are calculated for each of system module, based on equation (10), the next step is to decide which of the received values are going to be used as thresholds for assigning the modules to different classes. For this reason the received values are first sorted from lowest to highest. Then, it is decided how many percent of modules with the lowest values belongs to the first class and how many modules with the highest values belongs to the third class. Value of the importance factor calculated for a module which is the first one, according to the determined order, that belongs to the second class of modules constitutes the first threshold value. Value of the importance factor calculated for a module which is the last one, according to the determined order, that belongs to the second class of modules constitutes the second threshold value. If by δ_1 we denote the first threshold value and by δ_2 the second threshold value, then appropriate weight values for modules are determined by the formula (12).

$$w_i = \begin{cases} 1 \ when\ I_i < \delta_1 \\ 2 \ when\ \delta_1 \le I_i \le \delta_2 \\ 4 \ when\ I_i > \delta_2 \end{cases}, \qquad (12)$$

where:

$$i = \overline{1,n}$$

n – number of module in the system, $n \in N$.

Form of the reliability evaluation risk factor that is supposed to be used for the entire system is finally defined by formula (13). It is defined as weighted arithmetic mean of the individual risk factors of system modules.

$$H(k)=\frac{\sum_{i=1}^{n} w_i h_i(k)}{\sum_{i=1}^{n} w_i} \qquad (13)$$

The higher is value of the risk factor (13), the higher is risk that reliability predictions for the system under testing are not adequate to its actual reliability. To calculate value of the risk factor, data related to the modules modified in current development project are used. Obtained value is used in further analysis where, together with reliability evaluations produced by selected SRGM, as well as other crucial information concerning the system under testing, it is used to determine value of the maturity and readiness to integration indicator. By integration in this case it is meant that modules modified in a given individual development process are merged with the system being prepared for a customer.

4. Fault density adequacy factor

Software reliability evaluation produced by selected SRGM, together with information provided by the reliability evaluation risk factor, are based on only part of valuable information usually available for a decisive person during software validation process. From the model completeness perspective as well as to improve quality of produced outcomes, especially during early phase of the validation process, it seems to be reasonable to additionally use data related to software development process as such, data being a result of experts opinions and historical data concerning the system under testing. In software development process synthesis of various information that may be valuable from strategic decisions perspective, especially decision about delivery of a final product to a customer, is essential in today’s reality.

Fault density adequacy factor is supposed to reflect the level of fault density reached at a given moment in software validation process, in respect to the fault density requirements defined by a decisive person. Value of the factor is determined based on number of faults already detected in the software at a given moment in time, together with predicted number of faults that have been introduced into the system during code modification. The number of already detected faults is known at a given moment in time. The number of faults introduced into the system is determined by use of a Bayes net [10]. Structure of the used net is presented in figure 1.

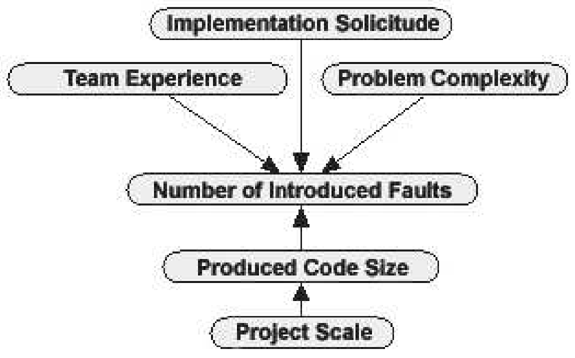

Fig. 1. Structure of a Bayes net used to evaluate number of faults introduced into the system

The number of faults introduced into the system is made dependent on selected information concerning the software development process. For a net node which is not descendant of any other node, according to the net structure, appropriate value can be specified by experts working in the development project which is being examined. Values for the remaining nodes are determined based on information provided by nodes which are parent nodes to a given node, according to the net structure. Allowed values for such a node are characterized by suitable probability distribution. Determined this way expected number of faults introduced into the system is used in further analysis.

At a given moment in the software validation process, the current fault density for the software under testing can be evaluated according to formula (14).

$$G_o = s\cdot\frac{b_o-b_w}{L}, \qquad (14)$$

where:
s – scale factor; represents number of source code lines used as a base for defining fault density requirement;
b_o – expected number of faults introduced into the system;
b_w – number of faults found until end of a given time step;
L – number of new or modified source code lines in current development project.

For the purpose of fault density evaluation an assumption is taken that the software is free of faults at the moment when development of new version of system starts. However, it is possible that during software validation process some legacy, previously unknown faults are revealed. That’s why if number of faults detected until end of a given time step is higher than the expected number of faults then fault density is assumed to be equal zero.

Having the software fault density evaluated it is then possible to determine value of the fault density adequacy factor $A_g(k)$. As a main reference the value of required fault density is used which, together with tolerance between value required and value achieved, is defined by a decisive person. The fault density adequacy factor takes values from range [0,1], with step which equals 0.1. Value 1 in this case means that achieved fault density is equal or lower than the required one. Based on the defined tolerance the subsequent fault density thresholds are determined. The defined tolerance reflects a ten percent threshold which, when crossed, results in value of the fault density adequacy factor decreased by 0.1, until it reaches 0. If by ρ we denote the required fault density and by τ the tolerance between value required and value achieved, then appropriate value of the fault density adequacy factor can be determined by the formula (15). Decided this way value of the factor is used later to determine value of the maturity and readiness to integration indicator.

$$A_g(k) = \begin{cases} 1 \quad when\ G_0 \le \rho + 0 \cdot \tau \\ 0.9\ when\ G_0 \le \rho + 1 \cdot \tau \\ 0.8\ when\ G_0 \le \rho + 2 \cdot \tau \\ \vdots \\ \vdots \\ 0.2\ when\ G_0 \le \rho + 8 \cdot \tau \\ 0.1\ when\ G_0 \le \rho + 9 \cdot \tau \\ 0 \quad when\ G_0 > \rho + 9 \cdot \tau \end{cases} \tag{15}$$

5. Maturity and readiness to integration indicator

Presented model introduces concept of a maturity and readiness to integration indicator which is supposed to be used as a guidance for a decisive person when decisions about integration of modified software modules with system that is being prepared for a customer are taken. Meaning of the indicator is directly related to development methodology applied for the system under testing. The way value for the indicator is determined is characterized by synthesis of information of different type, to finally provide a single value appropriate for a decisive process. In the presented model the decision that is supposed to be taken during validation process lies in the fact to agree or not agree on integration of modified software modules with system that is being prepared for a customer, based on the defined reliability objectives. By applying the presented approach the risk of taking wrong decision, that is decision to integrate modified modules while the software under testing has not reached appropriate level of its reliability, is reduced. It is assumed that reliability of examined software is a priority criterion for the decisive person.

The maturity and readiness to integration indicator expresses, by a percentage value, the level of fulfilling the requirement for software reliability defined by a decisive person. Value of the indicator for a given moment of software validation process is determined in the following way. The software reliability objective defined by a decisive person constitutes a level which, when reached, means 100% fulfillment of the requirement. First step is to compare software reliability evaluated by the selected SRGM, in case of this research by model Musa-Okumoto described in chapter 2, with level defined as the objective, to get preliminary level of the requirement fulfillment. For instance, when objective is set to 0.8 and evaluated reliability is 0.6 then the preliminary level is 75%. Then, it is assumed that the preliminary level can be treated as the final one when there are no symptoms showing that value of the evaluated software reliability might not be proper. By the symptoms the counted values of the reliability evaluation risk factor (13) and the fault density adequacy factor (15) are meant. It is assumed that when product $(1 - H(k)) \cdot A_g(k)$ equals 1 then the reliability evaluated by the selected SRGM is the final one and so the percentage value related to the evaluated reliability constitutes value of the maturity and readiness to integration indicator. Otherwise, that is when the above product is less than 1, appropriate percentage value to be used as the maturity and readiness to integration indicator value is counted by a proportional reduction in the percentage value corresponding to the evaluated software reliability. For instance, when the product gives 0.4 and previously counted preliminary level is 75% then the maturity and readiness to integration indicator gets value 30%.

6. Cost function

Taking into account economic side of software development project, in particular costs related to realization of software validation process and maintenance of the product on customer side, the proposed model introduces as well a cost function (16). The function allows to keep track of changes concerning overall costs for the ongoing project, taking into account real data available at a given moment in software validation process as well as predicted data concerning future. The cost aspect introduces additional limitation imposed on the main plan aiming to produce software with satisfying level of reliability but within decided budget. Therefore, together with quality analysis there is also cost analysis being performed for system under testing.

$$C(x|k) = C_f \cdot N(k) + C_h \cdot k + \\ + C_m \cdot (N(k+x) - N(k)) \,, \qquad (16)$$

where:
k – current time (corresponds to end of time step k);
x – additional time;
$N(k)$ – number of faults detected until end of time step k;
$N(k+x)$ – predicted number of faults until end of time $k+x$;
$C(x|k)$ – predicted overall cost until end of time $k+x$;
C_f – cost of single fault removal during software validation;
C_h – cost of conducting software validation during a single time step (k);
C_m – cost of single fault removal during software utilization by a customer.

Naturally, cost of fault removal when fault is detected by a customer is much higher than cost that needs to be incurred when fault removal takes place during software validation. It is assumed that cost of conducting the validation process as such is not negligible. By inclusion of this cost into analysis it is possible to judge whether to continue the validation process or not. It might be important especially in case when the quality goals have almost been met while budget limits are already or closely reached. The number of faults detected until end of time step k is known at a given moment when cost calculation takes place. The number of faults detected until additional time x passes is predicted by the selected SRGM. In case of this research it is model Musa-Okumoto described in chapter 2.

7. Reliability characteristics for software parallel validation

In case of developing a big scale system it is possible that many small production processes coexist and are realized, to the large extend, independently of each other by many development teams. Form the final product perspective it is however necessary to be able to monitor interesting characteristics for the main production process which aims to create a new version of the system.

Considering validation process which is being realized by a team involved in a single production process, two separable states can be defined, representing situation in which the process at a given moment in time can be. The first used state indicates searching for faults while the second one indicates fixing faults. This fact was used to describe a validation process together with relevant reliability characteristics. The following assumptions were taken:

- times between consecutive faults detection are independent random variables with the same probability distribution,
- faults detection as well as faults fixing intensity remain constant during examined time intervals and are the same for all independent validation processes of independent production processes; the intensity values are initially decided by domain experts, based on development teams characteristics, planned tasks characteristics, as well as development methodology; values of the intensities may be changed due to new evaluations based on fault detection and fault fixing information from the already finished time intervals,
- the number of testing teams is constant, equals the number of independent development processes, and each of the individual validation processes can be in one of two possible states – searching for faults or fixing a fault.

In case of individual validation process a state machine with two states is applied. Analysis of such process can be based on stochastic process theory, in particular theory of Markov process with continuous time and discrete set of states [11]. It was assumed that time of being in particular state of the process is characterized by exponential probability distribution with known value of parameter reflecting intensity with which the process leaves a state. Probability density for variables which represent time of being in fault detection state and time of being in fault fixing state are described by equations (17) and (18) respectively.

$$u(t) = \lambda e^{-\lambda t} \,, \qquad (17)$$

where:
λ – fault detection intensity in individual validation process.

$$u(t) = \lambda e^{-\lambda t} \tag{18}$$

where:

μ – fault fixing intensity in individual validation process.

From the main process perspective at a given moment in time some of the running in parallel individual processes are in fault detection state and some are in fault fixing state. Based on that fact it is possible to define a set of states for the main process, where state denoted as k means that currently k among all of the individual processes are in fault fixing state, whereas remaining *n-k* processes are in fault detection state. A Markov chain state diagram for the main process, which is a time-homogeneous Markov chain with finite space, is presented in figure 2.

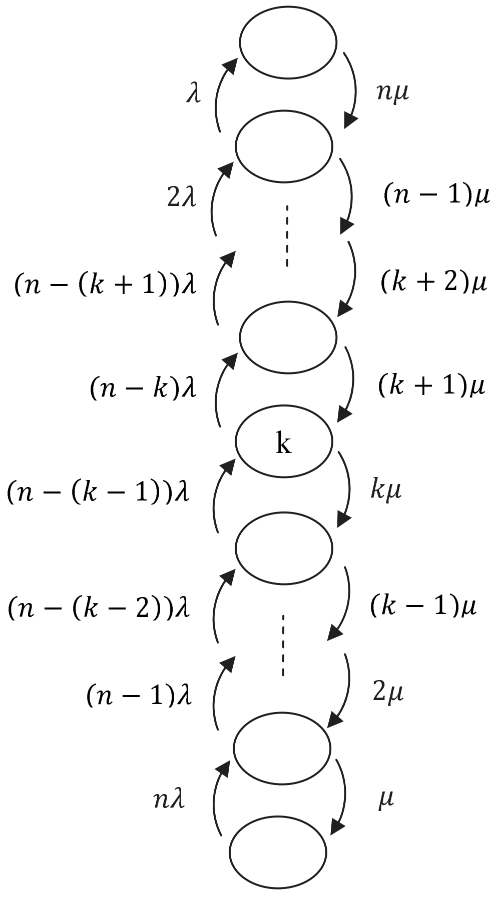

Fig. 2. Markov chain state diagram for the main process

A transition intensities between states in the main process are determined based on a feature of random variable distribution where the variable is defined as a minimum of independent exponentially distributed random variables. The feature tells that this way defined random variable is also characterized by exponential distribution with parameter which is a sum of parameters from distributions of the random variables which are arguments of the minimum function.

A time to leave a state in the main process depends only on the current state of the process and is characterized by exponential distribution with parameter dependent on the current state. Probability density for random variable X_k which represents time of being in state k of the main process is expressed by equation (19).

$$f(t,k) = (k\mu + (n-k)\lambda)e^{-(k\mu+(n-k)\lambda)t}, \tag{19}$$

where:

n – total number of individual processes constituting the main process;

k – number of individual processes being in fault fixing state;

$(n-k)$ – number of individual processes being in fault detection state;

λ – intensity of fault detection in individual validation process;

μ – intensity of fault fixing in individual validation process.

Value of probability of state k to state *k–1* transition and value of probability of state k to state *k+1* transition in the main process is determined based on equations (20) and (21) respectively.

$$q(k-1|k) = \frac{k\mu}{k\mu+(n-k)\lambda}, \tag{20}$$

$$q(k+1|k) = \frac{(n-k)\lambda}{k\mu+(n-k)\lambda}. \tag{21}$$

An unconditional probability of being in a given state of the main process is determined based on equation (22) which is a result of transformation of the Markov chain balance equation.

$$p(k) = \frac{\binom{n}{k}\left(\frac{\lambda}{\mu}\right)^k}{\left(1+\frac{\lambda}{\mu}\right)^n} \tag{22}$$

Time to find a fault when the main process is in a given state can be defined as a recursive relation (23). The equation incorporates value of time to find a fault for preceding state.

$$Y_k = X_k + q(k-1|k)Y_{k-1}, \tag{23}$$

where:

Y_k – random variable which expresses time to find a fault while the process is in state *k*;

X_k – random variable which expresses time of being in state *k*;

Y_{k-1} – random variable which expresses time to find a fault while the process is in state $k-1$.

Probability distribution for the Y_k variable can be determined by computing a convolution of probability densities of independent random variables which are components of the sum in equation (23). Based on Borel's convolution theorem, equation (24) is determined by applying Laplace transform and further simplifying transformations.

$$G^*(s,k) = \prod_{j=0}^{k} F^*\left(s \cdot \prod_{i=j+1}^{k} q(i-1|i), j\right), \tag{24}$$

where:
$G^*(\cdot)$ – Laplace transform of probability density function $g(\cdot)$ which expresses time to find a fault while the main process is in a given state;
$F^*(\cdot)$ – Laplace transform of probability density function $f(\cdot)$ which expresses time of being in a given state.

An absolute time to find a fault, irrespective of a state in which the main process currently resides, is determined based on equation (25).

$$Y = \sum_{k=0}^{n} p(k) \cdot Y_k \tag{25}$$

By applying Laplace transform for equation (25) and using equation (24) finally equation (26) is received.

$$K^*(s) = \prod_{k=0}^{n} \prod_{j=0}^{k} F^*\Big(s \cdot p(k) \cdot \cdot \prod_{i=j+1}^{k} q(i-1|i), j\Big) \tag{26}$$

The moment generation feature of Laplace transform or an inverse transform calculation can be used to get formulas that are used to determine expected value (27) and variance (28) of time to find next fault in the main process.

$$E(Y) = \sum_{k=0}^{n} p(k) \cdot \cdot \sum_{j=0}^{k} \prod_{i=j+1}^{k} q(i-1|i) \cdot E(X_j) \tag{27}$$

$$V(Y) = \sum_{k=0}^{n} p(k)^2 \cdot \cdot \sum_{j=0}^{k} \prod_{i=j+1}^{k} q(i-1|i)^2 \cdot V(X_j) \tag{28}$$

8. Model verification

Presented in previous chapters approach to software reliability verification during validation process was practically applied on data collected during realization of a real software development process which aimed to enhance functional capabilities of a complex, real-time system. Four independent teams were working on new version of the system, implementing separate functionalities. Therefore, from the proposed model perspective there were four individual production processes for which reliability analysis with use of maturity and readiness to integration indicator as well as cost function was performed. Also the method of determining reliability characteristics for a software under parallel testing performed by multiple teams was verified.

To get evaluations for Musa-Okumoto model parameters, current intensity of faults and expected number of software faults in a given time perspective, the SMERFS3 application was used as a tool which has proved its usefulness in software reliability researches [12].

Analysis of importance of system modules, necessary to determine weights of the modules and then the reliability evaluation risk factor value, shown that the applied complexity metrics in combination with functional criticality of a module are very effective in terms of ability to project the real situation in the evaluated system. Among forty seven modules, ten of them got weight 4, nine of them got weight 1 and the rest of modules got weight 2. Such a result also reflects reasonable architecture of the evaluated system in terms of program structure.

The length of a time step used during analysis was 24 hours. For each time step values of used factors were updated based on information concerning progress and results of the validation process. Picture 3 presents waveform of a function reflecting changes of values of the reliability evaluation risk factor over all 65 time steps for one of the individual processes, with two different values of the constant a from equation (8). Value 1,15 reflects quite low believe of a decisive person about the significance of deviation between number of faults expected and number of faults discovered, while value 1,5 reflects rather serious believe about such a fact. As can be seen from the picture, in the latter case the counted values of H(k) automatically expresses higher level of uncertainty about the evaluated software reliability and the wave is less linear due to longer period when the risk factor value reduces slowly.

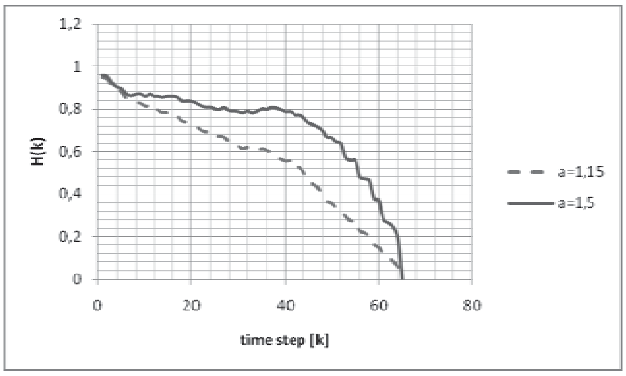

Fig. 3. Reliability evaluation risk factor value changes

The Netica application, being a tool which has proved its usefulness in researches requiring probability inference [13], was used to create the Bayes net presented in figure 1, in order to evaluate number of faults introduced into the software under testing. To built appropriate formulas to be used to determine values of parameters of probability distributions for the nodes "Produced Code Size" and "Number of Introduced Faults", relevant information from historical projects were used. Having the expected number of introduced software faults, the fault density value was updated after each time step and on this basis value of the fault density adequacy factor (15) was determined. Changes of value of the factor over all 65 time steps for one of the individual processes, called "process 4", is presented in figure 4.

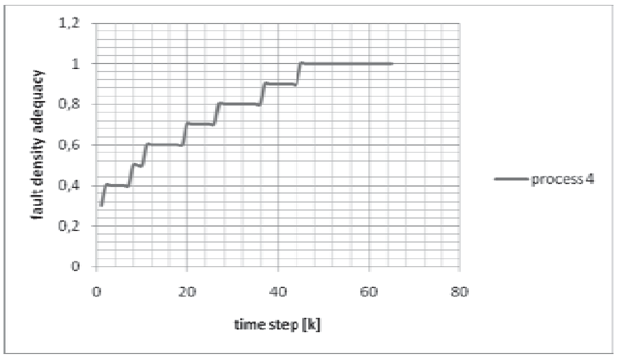

Fig. 4. Fault density adequacy factor value changes

Analysis aiming to compare results of applying into the decision making process the reliability evaluations produced by the selected SRGM and the indications given by the maturity and readiness to integration indicator was performed weekly. At the same time already incurred costs as well as forecasted costs were tracked, based on the cost function changes (16). Use of the maturity and readiness to integration indicator proved purposefulness of application of such a method under the circumstances when level of achieved software reliability is treated as a crucial criterion used by a decisive person to make key decisions in software development project. For one of the analyzed individual processes prolongation of the validation process until the maturity and readiness to integration factor reached satisfactory level, enabled to reveal by 43% higher number of faults in the software and to reduce by that the overall cost in the projected time horizon by almost 60%. It was assumed that the faults that could have been detected during the prolonged period of the validation process can be treated as a predicted number of faults from the formula (16) perspective, when additional time x is sufficiently long e.g. a year.

The method of reliability characteristics determination for the main process, presented in chapter 7, proved usefulness of the described approach.

Figure 5 presents changes, over all 65 time steps, of values of the probabilities of being in a particular state of the main process. The probabilities were determined based on formula (22). As there were four individual production processes, there were 5 states defined for the main process, according to principle from chapter 7. Necessary value of μ was determined by experts working for a project and it was constant along the validation process ($\mu = 0,00641$). Necessary value of λ was determined based on faults detection data and it was a subject to update before each of the planned reliability analysis sessions (weekly), see table 1.

Tab. 1. Values of parameter λ used for the consecutive reliability analysis weekly sessions

7	0.000989
14	0.000585
21	0.000378
28	0.000316
35	0.000229
42	0.00022
49	0.00021
56	0.000171
63	0.000131
65	0.000129

As it can be seen from figure 5, as time passes it is most probable that none of the individual processes is in fault fixing state, what is a logical consequence of elimination of consecutive faults from the software.

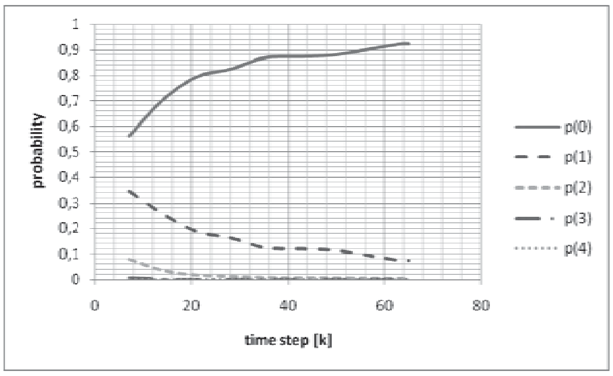

Fig. 5. Unconditional probability of being in particular state of the main process value changes

Figure 6 presents changes, over all 65 time steps, of value of the time to fault detection expected value E(Y), determined according to formula (27), including also value of the standard deviation D(Y), counted as a square root of the variance V(Y) determined according to formula (28). Values of $E(X_j)$ and $V(X_j)$ used in formulas (27) and (28) respectively, were counted based on the fact that random variable was assumed to be exponentially distributed. In such case, the expected value is represented by inverse of the rate parameter and the variance is represented by inverse of the rate parameter to the second power. Value of the rate parameter was a known value. As it was expected time to find next fault in the main process was successively longer and longer, reflecting the fact of continuous reduction of number of faults in the software.

By comparing received results of the time to fault detection expected values for the examined process E(Y), it turned out that applied method generates results very similar to approximation performed by model Musa-Okumoto, what is depicted in figure 7. Approximation by Musa-Okumoto is described as “M-O (aprox)”. The real observed times between consecutive faults detection in the main process is described as “TBF main process”.

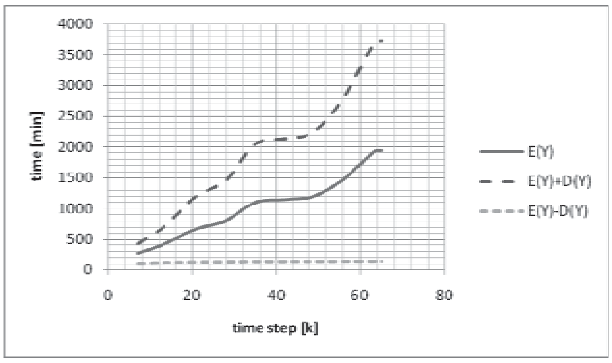

Fig. 6. Time to fault detection expected value in the main process value changes

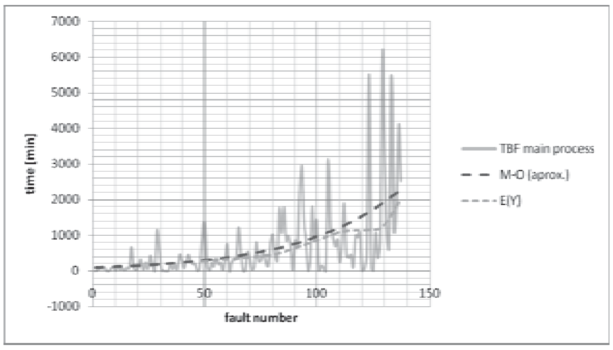

Fig. 7. Time between consecutive faults detection in the main process

9. Summary

Proposed model showed reasonable results of its practical applicability for the analyzed data set consisting of test related as well as system related information. Performed analysis shows also perspectives for potential improvements of the proposed approach to software reliability examination for complex systems. Additional attention shall be put on the results of application of the method described in chapter 7. As well, an attempt to develop more complex method for determining the maturity and readiness to integration indicator value is to be taken. That are the preferred directions for further researches on the field of software reliability examination.

10. Bibliography

[1] R. Pełka, “Software reliability growth models”, *Biuletyn Instytutu Systemów Informatycznych*, Nr 10, 19–29 (2012).

[2] M.R. Lyu, *Handbook of software reliability*, IEEE Computer Society Press, 1996.

[3] A.L. Goel, “Software Reliability Models: Assumptions, Limitations and Applicability”, *IEEE Transactions on Software Engineering*, 12 (1985).

[4] N.E. Fenton, “A Critique of Software Defect Prediction Models”, *IEEE Transactions on Software Engineering*, 5 (1999)

[5] J.D. Musa, K. Okumoto, “A Logarithmic Poisson Execution Time Model for Software Reliability Measurement”, *IEEE*, 1984.

[6] H. Pham, *System Software Reliability*, Springer-Verlag, 2006.

[7] S.H. Khan, *Metrics and Models in Software Quality Engineering*, Addison Wesley, 2002.

[8] T.J. McCabe, “A Complexity Measure”, *IEEE Transactions on Software Engineering*, 1976.

[9] S. Henry, D. Kafura, “Software Structure Metrics Based on Information Flow”, *IEEE Transactions on Software Engineering*, 1981.

[10] N. Fenton, M. Neil, *The use of Bayes and causal modeling in decision making, uncertainty and risk*, 2011.

[11] B. Korzan, *Procesy stochastyczne i teoria odnowy*, WAT, 1986.

[12] D.R. Wallace, “Practical Software Reliability Modeling”, *Software Engineering Workshop 200*, *Proceedings 26 Annual NASA Goddard*, NASA Goddard Space Flight Center, 2001.

[13] W. Cao, J. Ding, H. Wang, “Analysis of Sequence Flight Delay and Propagation Based on the Bayesian Networks”, *Fourth International Conference on Natural Computation*, ICNC 2008.

Wieloaspektowy model predykcji niezawodności oprogramowania w procesie testowania

R. PEŁKA

Badanie niezawodności oprogramowania stanowi istotną część realizacji planu jakościowego w procesie produkcji oprogramowania. Poprzez monitorowanie zmian wartości prognozowanej niezawodności oprogramowania w odniesieniu do założonych celów jakościowych można dokonywać analizy bieżącej sytuacji oraz w razie konieczności podejmować kroki sprzyjające realizacji założonego planu. Wykorzystanie w celu predykcji niezawodności jedynie modeli wzrostu niezawodności oprogramowania, bazujących na historii wykrywania błędów w badanym oprogramowaniu, wydaje się być podejściem zbyt uproszczonym. Podejście to w pewnych okolicznościach realizacji procesu walidacji oprogramowania może być obarczone dużym błędem i wpływać na podejmowanie błędnych decyzji przez decydenta. W związku z tym, w zaproponowanym modelu wykorzystuje się szereg dodatkowych informacji o testowanym oprogramowaniu oraz samym procesie walidacji w celu uzyskania bardziej wiarygodnych efektów analizy niezawodnościowej, będących jednocześnie odpowiednią informacją zwrotną dla decydenta z punktu widzenia założonych realiów prowadzenia projektu programistycznego. Integralną część prezentowanego podejścia stanowi aspekt wyznaczania charakterystyk niezawodnościowych systemu testowanego równolegle przez kilka niezależnych zespołów.

Słowa kluczowe: niezawodność, oprogramowanie, modelowanie, testowanie.