International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org Volume 3 Issue 7‖ July 2014 ‖ PP.01-06

Software Reliability Assessment Using High-Order Markov Chains

Vitaliy Yakovyna, Dmytro Fedasyuk, Oksana Nytrebych, Iurii Parfenyuk, Volodymyr Matselyukh
Software department, Lviv Polytechnic National University, Ukraine

ABSTRACT : *The assumption of independent execution of the component in the software reliability prediction models using component-based approach is a simplification of the real work of the software. In this paper the method of software reliability prediction that takes into account dependencies between software components is described. It uses higher-order Markov processes and consists of the following steps: definition of each component's failure rate, probability of transitions between components, the optimal order of Markov process and computation of the whole software system reliability itself.*

KEYWORDS : *higher-order Markov chain, software usage model, transition probability matrix, failure intensity, software reliability.*

I. INTRODUCTION

Nowadays the impact of computer system failures on society has different consequences – from minor inconveniences (malfunctioning appliances) to the loss of human life (a refusal of flight control system or a failure in the medical software). Therefore, it is important to determine the software reliability that not only provides customers more confidence in the product, but also can reduce the cost of testing. In this paper, according to [1], reliability means the probability of software execution during specified period of time under particular circumstances without any failures. Many models for the software reliability analysis exist nowadays. They are usually split into "white box" and "black box" models [2] depending on the use of information about the architecture of software. It is clear that the "white box" model can describe the software reliability more adequately as they evaluate the internal structure of software and interaction of its components. Therefore this type of model is also called differently – models which are based on an architectural approach. These models are divided into additive models, path-based and component-based models [3].

Component-based approach for software reliability assessment and prediction using Control Flow graph to describe the software architecture [2, 3, 4] is the most common. Analysis of the reliability of this approach is done using software Markov usage model [5, 6], which allows to avoid general assumptions about failure intensity distributions and incorporate both the operational profile and test coverage explicitly and automatically into reliability. Disadvantage of models which use component-based approach is usage of the first-order Markov chain for software reliability modeling [1-4]. Such models simply don't take into account the interdependence of software components execution, which is a usual case in real software systems.

To count the mutual dependency of components execution in software usage model it is proposed to use higher-order Markov chains (HOMC). This paper describes a method of reliability analysis of software based on higher order Markov chains in order to improve the adequacy of software reliability prediction.

II. MODEL OF SOFTWARE RELIABILITY EVALUATION BASED ON HIGHER ORDER MARKOV CHAINS

As mentioned, the usage of higher order Markov process will allow to assess the software reliability more accurately. Based on the Gochale's evaluation model of software reliability [7], the reliability of the whole system is calculated as:

$$R = \prod_{l=1}^{n} R_l \ . \qquad (1)$$

R_l – reliability of each component, n – number of software components. Use of higher-order Markov process (let N - the order of the model) in this model will result in next definition of each component's reliability:

$$R_l = e^{-\int_0^{V_{ij\ldots kl}\, t_{ij\ldots kl}} \lambda_l(t)\,dt}, \quad (2)$$

To find $V_{ij..kl}$ - the expected number of visits to component *l* depending on execution of previous *N* components, a system of linear equations must be solved:

$$V_{j..kl} = q_{j..kl} + \sum_{i=1}^{n-1} V_{ij\ldots k}\, p_{ij\ldots kl}, \quad (3)$$

$p_{ij\ldots kl}$ - probability of transition to component *l* depending on execution of previous *N* components; $q_{ij\ldots k}$ - initial probability vector; $t_{ij\ldots kl}$ - execution time of component *l* which depends on execution of previous *N* components.

Given the numerical values of all model parameters, the reliability of each component using formula (2) and the value of reliability of the whole software system (1) can be calculated. This model is hierarchical, because initially architectural model parameters are calculated and then the behavior of failures of each component is taken into account to assess the software reliability.

Obviously the usage of HOMC introduces other important subtasks to be solved:

- Definition of failure rate $\lambda_i(t)$ of each software component;
- Definition of the transition probability matrix;
- Determination of the optimal order of Markov chain.
- Approaches to resolve these issues are presented in the next sections.

III. DETERMINATION OF THE FAILURE INTENSITY OF SOFTWARE COMPONENTS

Famous software reliability models can be used to calculate the failure rate of each component. Most of these models are based on a nonhomogeneous Poisson process. Many existing models of software reliability can be described within the inhomogeneous Poisson process [8-9]. Numerous empirical researches confirm the validity of the models of this type [10]. Some models of the inhomogeneous Poisson process describe the exponential increase of reliability, while others show the S-shaped growth, depending on the nature of the phenomenon of reliability growth during testing [11]. The appearance of S-curve is explained with the various factors and literature contains sufficient amount of models based on this form. In literature references *S*-like growth is associated with different reasons: the interdependence of software defects [11], less efficient testing on initial stages in comparison with following [12] and so on. To increase the degree of adequacy of existing software reliability models based on a nonhomogeneous Poisson process and to take into account the impact of model complexity on the behavior of software failure flow parameter a model of software reliability with complexity index [13] is constructed and analyzed. In this model, flow failures parameter function is as below:

$$\lambda_i(t) = \alpha_i \beta_i^{s+1} t^s \exp(-\beta_i t), \quad (4)$$

α_i – coefficient which shows general amount of failures in the *i*-th software component; β_i – coefficient which shows general duration of testing process in the *i*-th software component, *s*- software complexity index, which generalizes *S*-shaped models.

Obviously, for calculation $\lambda_i(t)$ of each component using this model it is necessary to make a test of software system to obtain statistics of failures and their distribution in time. For higher data accuracy testing should be performed with the maximum code coverage, which will allow identifying more failures. To ensure this condition, the method of automated test scripts construction using either "white" or "black box" strategies [14] based on a new variable state-based software usage model based on its variables [15], has been developed.

Variable state-based software usage model, which is an input parameter for the automated tests construction, is shown as a directed graph $G = \{S, P\}$, where $S = \{S^0, S^1, \ldots, S^n\}$ – set of software components S^i, P – set of transitions between components. Each node of graph S^i is the set $\{V^i_{used}, V^i_{change}, V^i_{error}\}$, where V^i_{used} – set of variables and the corresponding equivalence classes used in the

component S^i, list of variables that change in the component S^i; V^i_{change} – list of variables that change in the component S^i (this set is the union of the set of variables that can be changed by the user and tested with "black box" method, and set of variables that can be changed only by the internal logic of the program and tested only with "white" box testing); V^i_{error} – the list of variables and corresponding incorrect equivalence classes, which can cause failures in the software system. The importance of consideration of the usage of the software variables during usage model construction, and construction of test cases based on this model, follows from the fact that many software reliability analysis models use software metrics, which in turn take into account the use of software variables.

It should be noted that the proposed test generation method [14] is iterative: firstly test suite without information about the failure is constructed, and then after this set run on the software and getting the information on detected failure, the next set of tests is built until failure will not be fixed. This method of automated construction software test cases provides uniform code coverage and improves the efficiency of the testing process, reducing the time, financial and human resources.

IV. DETERMINATION OF TRANSITION PROBABILITIES BETWEEN COMPONENTS

Determination of transition probabilities between components is a difficult task today for implementation. One way of solving this problem is approach, based on results of software execution monitoring [16], since during software execution control is passed between components, and having the results of the transmission management the transition rates between the components can be determined. To determine the transition probabilities to the component C_j from C_i it is necessary to determine the ratio of the number of transitions from C_i to C_j component to the total number of transitions from component C_i. To test the proposed approach, the authors developed logger on Java. Designed logger lets to explore the software without making any changes to the code. This approach makes it possible to define and refine the matrix of transition probabilities between software components even at the stage of implementation and actual usage, with consideration of real software usage scenarios by different classes of users. In addition to software reliability prediction on the early stages of the software life cycle, which allows avoiding costly fixing of failures on the later stages, determination of transition probabilities matrix can be done by analyzing the UML diagrams [17]. Transition probabilities matrix, more precisely its structure, can be created from the class diagram, but it should be noted that it is impossible to obtain transition probabilities and time spent in the components. Using use-case diagrams based on all use cases execution probabilities (or relative frequencies) the probability of execution of software components and transitions between them can be estimated. Besides, using UML Sequence diagram, we can get the number of transitions between the components and then easily determine the transition probabilities matrix between them and the time spent in the component. After analyzing UML Sequence Diagrams developed during the software design it is required to calculate the total number of transitions between components and form a transition probabilities matrix according to the approach described above [16]. But it must be remembered that at this stage of the life cycle (design phase) software components design can only be approximated, and mentioned probabilities are prior and based on expert estimates.

Another way to obtain a matrix of probabilities of transitions between the components is the use of hidden Markov chains. As a result of their capabilities, hidden Markov models (HMMs) become more and more frequently used in modeling. Examples include, eg, the detection of intrusion into software systems, fault diagnosis, network traffic modeling, estimation and control, speech recognition, part-of-speech tagging and genetic sequence analysis applications. There are three fundamental questions we might ask of an HMM. What is the probability of an observed sequence? What is the most likely series of states to generate the observations. And how can we learn values for the HMM's parameters *P* and *B* given some data? Where *P* is transitions probabilities matrix and matrix B encodes the probability of hidden states generating output υ_k given that the state at the corresponding time was s_j [18]. Last and the most needed for us question is reached by using Baum-Welch algorithm [19]. Notes on the Baum-Welch algorithm. Initial point for the Baum-Welch algorithm is a completely initialized HMM. This means that the number of states, number of observation symbols, transition probabilities, initial probabilities and observation probabilities need to be defined. The algorithm then iteratively improves the model's parameters $\lambda = (P, B, q)$ until a (local) maximum in sequence likelihood is reached. In each *M*-step, the expectation values of the previous *E*-step are used and vice versa. Several properties of the algorithm can be derived from that:

- The number of states and the size of the alphabet are not changed by the algorithm.

- The model structure is not altered during the training process: if the reis no transition from state s_i to s_j ($p_{ij} = 0$), the Baum-Welch algorithm will never change this.
- Initialization should exploit as much a-priori knowledge as possible. If this is not possible, random initialization can be used [20].

Therefore, to get transition probabilities matrix we need to switch to the hidden Markov model where execution of one or other system component can be a hidden state, and possible observations - stop responding, or the issuance program failures, system crashes at a certain stage of its implementation. To run the Baum-Welsh algorithm, we need to initialize the transition matrix, the matrix of observations and initial state vector of hidden Markov models and it need to have a sequence of system observations. On the output of the algorithm the initial probability vector *q* and the transition probabilities matrix *P*, which can then be used in the district (3) to calculate the expected number of visits to component *i* depending on visits to previous *N* components, are received.

V. DETERMINATION OF MARKOV CHAIN ORDER

Another problem to be solved in order to apply the higher-order Markov chains in the analysis of software reliability is to determine the optimal order of Markov process, which depends on the software system under research, in other words on the modeled object.

To determine the optimal order of Markov processes in the software reliability assessment tasks it is proposed to use information criteria, namely criteria with penalty for complexity, because they are not tests of hypothesis and do not use the significance level [21]. In [22] cases of application of these criteria based on the number of observed data samples are shown.

Thus, when the size of empirical data set is small, BIC criterion should be used for the software reliability behavior assessment, since it uses stronger penalties even when the sample size $n > 8$ $(\ln(n)\, k > 2k)$, which should allow to avoid unwarranted increase in the optimal model order due to insufficient empirical data and increases the probability of selecting "exact" model (assuming the existence of such model). When the set of empirical data is large, criteria selection question remains open and requires experimental researches as AIC on the one hand does not provide for "exact" model, and estimates the degree of proximity of the set of test patterns to the "true" model, on the other hand the criterion BIC increases the probability of selecting "exact" model (assuming the existence of the such model) with set size growth.

VI. ESTIMATION OF SOFTWARE RELIABILITY BASED ON HIGHER ORDER MARKOV CHAINS

Thus, the process of software reliability analysis using HOMC that allows to count the mutual dependencies between component executions and respectively model the process of software reliability analysis more adequately, consists of the following two steps:

1. Definition of the input parameters of the model for evaluating the software reliability, that in turn has a number of sub-tasks;

1.1. Determination of the failure rate of each component using "black box" models (proposed to use software reliability model with a dynamic index of project complexity);

1.2. Calculation the transition probabilities matrix and components execution time (using the logger, UML diagrams, Hidden Markov Process).

1.3. Obtaining of optimal order Markov process (depending on the size of the data sample used criteria for AIC or BIC-family).

2. Evaluation of software reliability using MCHO.

Generalized scheme of this method is presented in Figure 1.

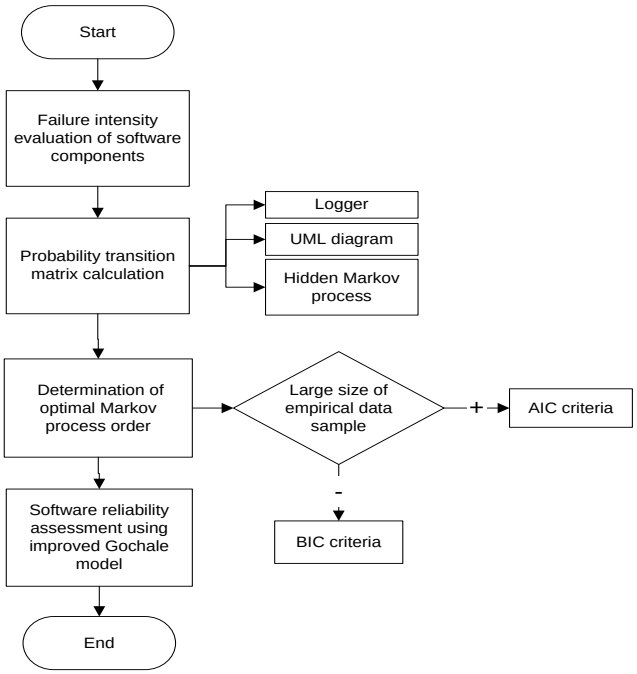

Fig. 1. Schema of method for software reliability analysis based on HOMC.

VII. CONCLUSION

The article describes the usage of higher order Markov chains for the problem of software reliability modeling that allows consideration of the mutual dependencies in software components execution, as due to the complexity of modern software architecture and the set of it's usage scenarios, the assumption of independence of the software component execution is not always true. As a result - the method for the analysis of software reliability based on HOMC has been proposed and all of its subtasks with possible solutions have been described. Namely, the software reliability model with a dynamic index of project complexity is considered to obtain failure rate of each component; an example of the logger, UML diagrams, Hidden Markov Process to calculate the transition probabilities matrix have been provided; depending on the size of the data set criteria of AIC or BIC-family are being used to determine the optimal order Markov process; software reliability estimation using HOMC.

REFERENCES

[1]. W. Burkhart, Z. Fatiha, *Testing Software and Systems* (23rd Ifip Wg 6.1 International Conference, 2011, 236).
[2]. K. Goseva-Popstojanova, A.P. Mathur, K.S. Trivedi, Comparison of architecture–based software reliability models,*12th International Symposium on Software Reliability Engineering*,2001, 22-31.
[3]. K. Goševa-Popstojanova, S. Trivedi, Architecture-based approach to reliability assessment of software systems, *Performance Evaluation*,4, 2001,179-204.
[4]. Thanh-Trung Pham, Xavier Defago , Reliability Prediction for Component-based Systems: Incorporating Error Propagation Analysis and Different Execution Models,*12th International Conference on Quality Software*, 2012, 106-115.
[5]. Heiko Koziolek, Operational Profiles for Software Reliability, *Dependability Engineering*,2, 2006, 119-142.
[6]. S. Jungmayr, J. Stumpe, Another motivation for usage models: generation of user documentation, *CONQUEST'98, Nüremberg, Germany*, 1998.
[7]. S. Gokhale, W. Wong, J. Horgan, S. Kishor, An analytical approach to architecture-based software performance reliability prediction, *Performance Evaluation*, 58(4), 2004, 391–412.
[8]. A.L. Goel, K. Okumoto, Time-Dependent Error-Detection Rate Model for Software and other Performance Measures , *IEEE Transactions on Reliability*, 28(3), 1979, 206–211.
[9]. J.D. Musa, A theory of software reliability and its application, *IEEE Transactions on Software Engineering*, 1(3), 1975, 312–327.
[10]. A. Wood, Predicting software reliability, *Computer,* 29 (11), 1996, 69-77.
[11]. S. Yamada, M. Ohba, S. Osaki, S-shaped reliability growth modeling for software error detection, *IEEE Transactions on Reliability*, 32(5), 1983, 475–478.
[12]. C. Rahmani, A. Azadmanesh, *Exploitation of Quantitative Approaches to Software reliability* (University of Nebraska at Omaha, 2008, 32).
[13]. Y. Chabanyuk, V. Yakovyna, D. Fedasyuk, M. Seniv, U. Himka, Construction and research of software reliability model with project size index, *Software Engineering*, 1, 2010, 24–29. (in Ukrainian)
[14]. D. Fedasyuk, V. Yakovyna, P. Serdyuk, O. Nytrebych, The method of software test case construction based on the analysis of its variables, *Information Technology and Computer Engineering*, 2014 (in Ukrainian, to be published).
[15]. D. Fedasyuk, V. Yakovyna, P. Serdyuk, O. Nytrebych, Variable state-based software usage-model based on its variables, *Econtechmod : an international quarterly journal on economics in technology, new technologies and modelling processes*, 2014 (to be published).

[16]. V. Yakovyna, I.Parfeniuk, Determination of transition probabilities between software components, written in java, based on monitoring of its execution, *Proceedings of the XIIth International Conference "The Experience of Designing and Application of CAD Systems in Microelectronics"*, 2013, p. 382.
[17]. V. Yakovyna, I. Parfeniuk, Evaluation matrix transition probabilities between software components based on UML Use Case Diagram, *Proceedings of the IXth International Conference PERSPECTIVE TECHNOLOGIES AND METHODS IN MEMS DESIGN*, 2013, p. 421.
[18]. D. Ramage, Hidden Markov Models Fundamentals, *Stanford University CS229 Section Notes*, 2007, 1-13.
[19]. S. Tu, Derivation of Baum-Welch Algorithm for Hidden Markov Models, http://people.csail.mit.edu/stephentu/writeups/hmm-baum-welch-derivation.pdf.
[20]. Felix Salfner. Event-based Failure Prediction: An Extended Hidden Markov Model Approach. dissertation.de - Verlag im Internet GmbH, Berlin, Germany, 2008. (Available as PDF).
[21]. P.Burnham, D. Anderson, *Model Selection and Multi model Inference: A Practical Information-Theoretic Approach* (Springer, 2002, 488).
[22]. V. Yakovyna, D. Fedasyuk, O.Nytrebych, The analysis of information criteria usage in software reliability assessment models, *Proceedings of the National Technical University "KPI"*, 2014, 108-115. (in Ukrainian)