

# Discrete and Continuous Time High-Order Markov Models for Software Reliability Assessment

Vitaliy Yakovyna and Oksana Nytrebych

Software Department, Lviv Polytechnic National University, Lviv, Ukraine  
vitaliy.s.yakovyna@lpnu.ua, ksenija.volynj@gmail.com

**Abstract.** Due to the critical challenges and complexity of modern software systems developed over the last decade, there has arisen an ever increasing attention to look for products with high reliability at reasonable costs. Software development process moves toward component-based design, and architecture based approach in software reliability modeling is widely used. However, in lots of models for software reliability assessment the assumption of independent software runs is a simplification of real software behaviour. This paper describes two software reliability models that use high-order Markov chains thus taking into account dependencies among software component runs for more accurate software reliability assessment. The efficiency and accuracy of developed models is investigated by the example of several software products. It is shown that using the software reliability models based on the high-order Markov chains results in the software reliability assessment accuracy up to 10–20%.

**Keywords.** Software reliability, architecture-based model, high-order Markov chains.

**Key Terms.** SoftwareComponent, SoftwareSystem, MathematicalModel.

## 1 Introduction

Computer systems are widely used in modern industry for control and automation purposes. All of these systems are controlled by software. Thus, software is used in air traffic control, nuclear power plants, automated patients monitoring etc. Therefore high requirements for software reliability are demanded because failures of such systems can lead not only to significant financial losses, but also threaten human life and health. Although techniques that allow assessing and ensuring the specified hardware reliability requirements have been developed, there are no common approaches for software systems assessment.

According to STD-729, software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1, 2]. Although Software Reliability is defined as a probabilistic function, and comes with the notion of time, it should be noted that, contrast to traditional hardware relia-

bility, software reliability is not a direct function of time [1]. Electronic and mechanical parts may become "old" and wear out with time and usage, but software will not rust or wear-out during its life cycle. Software will not change over time unless intentionally changed or upgraded.

The history of software reliability assessment methods and tools began in the 60s of last century. A number of researchers [3-7] worked on issues of development and research of software reliability analysis and assessment models and methods that would allow reducing the costs required for software testing stage. New software reliability assessment models that reflect the internal structure of the application and interaction of its components [6], called architectural-based models, are being developed because of increasing complexity of software systems and the result of expanding their functional purpose.

In known models based on architectural approach [6, 7] the theory of first order Markov chains is used with the assumption that the software components runs are independent. This assumption is not always true due to the complexity of modern software architecture and huge set of usage scenarios this assumption [8, 9]. Therefore, for the development of adequate software reliability assessment models, which will improve the testing process (e.g. allow to reduce the needed resources), one should consider the high-order Markov chains, which allow to take into account the interdependence of components runs [10].

This paper describes the software reliability assessment models based on architectural approach, which use discrete and continuous time high-order Markov chains and their comparison based on real world data. Nowadays architecture-based software reliability models have been well studied in theory, but there is lack of papers describing their practical applications. Developing of high-order models for software reliability assessment is not described enough in literature as well. Thus, the development of new architecture-based software reliability assessment models that take into account the dependencies among software component runs along with their application to real world data is still a problem waiting for solution.

## 2 Software Reliability Model with Discrete Time High-Order Markov Chain

This model considers absorbing Markov process that implies the existence of one or more absorbing states, i.e. states that, once entered, cannot be left.

The developed software reliability assessment model [10] is hierarchical, that is initially software architecture parameters are calculated based on software usage model [11] using the theory of Markov processes, and then the behaviour of each component failures is taken into account.

The discrete time HOMC software reliability model can be described using the following components –  $\{C_i\}$  is the graph with vertices corresponding to software components, while edges indicates the program control flow ( $i = \overline{1, N}$ , where  $N$  is the number of program components);  $\mathbf{P} = \{p_{ij..kl}\}$  is the high-order transition proba-

bility matrix ( $p_{ij..kl}$  – transition probability from component  $i$  to component  $l$  depending on being in previous  $K$  components);  $\mathbf{Q} = \{q_{ij..kl}\}$  is the initial probability vector;  $\lambda_i(t)$  is the failure rate of  $i$ -th software component.

According to this model the reliability of the whole system is calculated as

$$R = \prod_{l=1}^N R_l \quad (1)$$

In turn, the reliabilities of each component ( $R_l$ ) using high-order Markov chains are calculated by

$$R_l = \exp\left(-\int_0^{\sum_{j..k} V_{j..kl} t_{j..kl}} \lambda_l(t) dt\right) \quad (2)$$

To calculate  $V_{j..kl}$  – the expected number of visits a component  $l$  depending on being in previous  $K$  components – one has to solve the following system of linear equations:

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

here  $t_{ij..kl}$  denotes the time spent at the component  $l$  depending on being in previous  $K$  components.

### 3 Software Reliability Model with Continuous Time High-Order Markov Chain

Using discrete-time model has a number of significant simplifications and restrictions. Thus, this model takes into account only the number of visits to  $i$ -th component without taking into account of the distribution function of this random variable (while taking it into account the expected value should be used). In addition, it is clear that at any given time  $t$  the software failure is generally caused by the failure of the software component, which is executed at a given time (in case of sequential connection of software components in Reliability Block Diagram). Thus, to increase the degree of adequacy of the software reliability architectural model the continuous time Markov chains should be used.

The continuous time HOMC software reliability model can be described using the following components –  $\{C_i\}$  is the graph with vertices corresponding to software components, while edges indicates the program control flow ( $i = \overline{1, N}$ , where  $N$  is the number of program components);  $\mathbf{A} = \{a_{ij}\}$  is the high-order transition probabil-

ity matrix  $i, j = \overline{1, N}$  (the values of matrix elements  $a_{ij}$  depends on the way of getting into the state  $i$ );  $\mathbf{P} = \{p_i(t)\}$  is the probability vector, where  $p_i(t)$  is the probability being in state  $C_i$  at time  $t$ ;  $\lambda_i(t)$  is the failure rate of  $i$ -th software component.

In this case, the failure rate of a software system consisting of  $N$  components can be written as [12]

$$\lambda(t) = \sum_{i=1}^N p_i(t) \cdot \lambda_i(t) \quad (4)$$

here  $\lambda_i(t)$  is the failure rate of  $i$ -th component,  $p_i(t)$  is the probability of  $i$ -th component execution at time  $t$ .

Components failure rates  $\lambda_i(t)$  can be obtained from the results of unit testing using known software reliability models, for example ones based on an inhomogeneous Poisson process [13].

If the flow control between the components of the software is presented as a Markov process with continuous time, assuming that the  $i$ -th state of the process is the execution of  $i$ -th component, the time dependences of the probabilities being in  $i$ -th state ( $p_i(t)$ ) can be obtained by solving the system of equations of the Kolmogorov–Chapman [14] for this process:

$$\frac{dp_i(t)}{dt} = -\sum_{j \in S} \omega_{ij}(t) p_j(t) + \sum_{j \in S} \omega_{ji}(t) p_j(t), i \in S \quad (5)$$

here  $\omega_{ij}(t)$  is the transition intensity from component  $i$  to component  $j$  at time  $t$ , and  $S$  denotes the set of all system states. In general the transition intensities  $\omega_{ij}(t)$  depends on the transition probabilities  $a_{ij}$  and could be calculated from latter.

To take into account the interdependence of execution of software component (and consequently changes of transition probability from  $i$ -th state) depending on the way of getting to the current state, the high-order Markov chain should be used (the order  $K$  of the model determines the accounted length of the path). In a case of high-order Markov chains the actual problem is to calculate the transition probabilities depending on the program control flow background.

It is well known that a high order Markov chain can be represented as a first order chain by appropriate redefining the state space [15]. For software implementation of models that use high-order chains, it is necessary to have the formalized algorithm for this representation. Using the analogy with the known Erlang phase method [16] a high-order Markov process can be represented as an equivalent first-order process with additional virtual states. Each state of the original graph  $\{C_i\}$  (geometric dia-

gram, showing the possible states of the system and the possible transitions of the system from one state to another) is split into such number of virtual states as many different paths of length  $K$  to this state exist. Thus the problem solution essentially is reduced to using of graph theory. Therefore, to calculate the number  $m_{ij}^K$  of chains of  $K$ -th order from state  $i$  to state  $j$  the following expression based on the Floyd method can be used:

$$m_{ij}^K = \sum_{j=1}^S (m_{il}^{(K-1)} + e_{il}) m_{lj}^1 \quad (6)$$

here  $e_{ij}$  are the elements of the identity matrix.

Using (6) one can build an equivalent graph  $\{C'_i\}$  which is a representation of the initial graph  $\{C_i\}$ , taking into account all  $K$ -th order paths to  $i$ -th state. This allows to avoid the dependencies of the transition intensities  $\omega_{ij}(t)$  on the program control flow background. Then the time dependence of the software component execution probability is obtained by solving the system of Kolmogorov–Chapman equations (5) for the equivalent first order Markov process. Using this dependence in (4) together with the component failure rate, the failure rate of the whole software system can be calculated.

For calculation of software reliability measures, based on the obtained relationship (4), one can use the following relations [14]:

- reliability function  $P(t)$  – the probability that no failure will occur within the time interval  $[0, t]$  – can be calculated as

$$P(t) = \exp\left(-\int_0^t \lambda(\tau) d\tau\right) \quad (7)$$

- mean operating time to failure  $T_1$

$$T_1 = \int_0^{\infty} P(\tau) d\tau \quad (8)$$

#### 4 Determination of the Markov Chain Order for Software Reliability Assessment

In this paper it is proposed to use AIC and BIC criteria [17] since they are not a hypothesis test and they don't use the significance level. Although these criteria give consistent results and don't depend on the estimated model order. They are efficiently

used for weather forecasting and selection of adequate environmental model [18]. But the usage of these criteria in software reliability modeling still remains unexamined.

In general AIC criterion [19] is calculated by the following expression:

$$AIC = 2k - 2\ln(L) \quad (9)$$

here  $k$  is the number of independent parameters in the model, and  $L$  is the model's maximum likelihood function.

If the value of this software model's maximum likelihood function is substituted in the expression (9), and instead of parameter  $k$  one substitute the number of the  $K$ -th order model parameters, which contain  $N$  components ( $k = N^K(N-1)$ ), then the following expression can be obtained [20]

$$AIC(K) = 2N^K(N-1) - 2\ln\left(\prod_{i,j,\dots,k,l} p_{ij..kl}^{n_{ij..kl}}\right), \quad (10)$$

here  $n_{ij..kl}$  is the number of transitions from component  $i$  to component  $l$  depending on being in previous components ( $i \rightarrow j \rightarrow \dots \rightarrow k \rightarrow l$ ) in observed sequence and  $p_{ij..kl}$  is the transition probability in this sequence.

As it can be seen from (10), this criterion is independent of sample size. Therefore, AIC criterion is used in the case of a large sample size (observable sequence) when the software is tested many times and the sequence of software component runs is logged.

An alternative to AIC criterion is BIC [21], which takes into account the number of elements in observable sequence and expressed as

$$BIC(K) = -2\ln(L) + \ln(n)k. \quad (11)$$

In the case of using the BIC criterion for the Markov chain optimal order determination in the case of software reliability modeling, the expression similar to AIC criterion (10) is obtained:

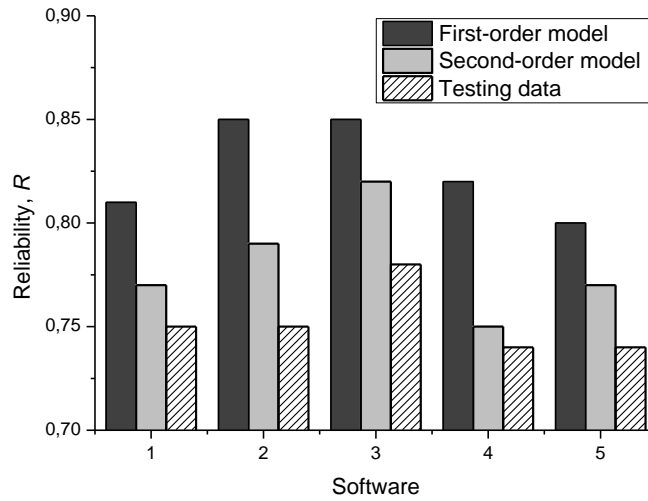
$$BIC(K) = 2N^K(N-1)\ln(n) - 2 \sum_{i,j,\dots,k,l} n_{ij..kl} \ln p_{ij..kl}. \quad (12)$$

It is worth noted that the BIC criterion should be used at the small sample size of empirical data concerning the software usage, because it imposes stronger penalties even when the sample size  $n > 8$ .

Thus, after determining the order of Markov chain, the well-known classical Markov tools can be used for software reliability assessment as it was described above.

## 5 Verification of the Models

To study the efficiency of the developed software reliability assessment model that uses the discrete time high-order Markov chains, the reliabilities of the five software systems, developed by one of the authors, were calculated using the first and the high order Markov chain models. The average amount of software components in tested software systems is 10. The results of reliability calculations using the first and the high-order models are shown in Fig. 1 along with the reliability obtained from unit testing data.



**Fig.1.** The comparison diagram of the software reliability assessment accuracy using first and high order discrete time models.

As shown in fig. 1, the usage of software reliability assessment model based on discrete time high-order Markov chains makes it possible to increase the software reliability accuracy to 6% even for software systems with a small number of components.

To illustrate the efficiency of continuous time software reliability model, the reliability of the authors' developed program with four components (in this case, each component refers to one of the following program classes – Input, Calculation, Output, Exit) was analyzed. The values of each component failures rates are presented in Table 1.

**Table 1.** Failure detection frequency for each component

Component	Failures detection frequency
<i>Input</i>	0.11
<i>Calculation</i>	0.18
<i>Output</i>	0.09
<i>Exit</i>	0.01

The probability transition matrix and the initial probability vector of the Markov chain were calculated and are summarized in Table 2 and Table 3 correspondingly.

**Table 2.** The first order probability transition matrix

<b>Component</b>	<i>Input</i>	<i>Calculation</i>	<i>Output</i>	<i>Exit</i>
<i>Input</i>	0.2987	0.66233	0	0.03897
<i>Calculation</i>	0.2666	0.05	0.6333	0.0501
<i>Output</i>	0.5	0	0.3148	0.1852
<i>Exit</i>	0	0	0	1

**Table 3.** The first order initial probability vector

<b>Components</b>	<i>Input</i>	<i>Calculation</i>	<i>Output</i>	<i>Exit</i>
<i>Probability</i>	1	0	0	0

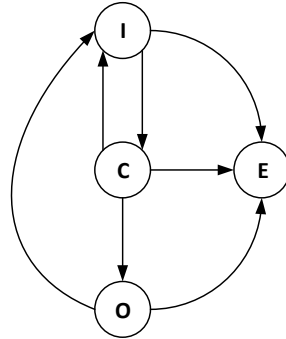
The AIC criterion was used for optimal Markov chain order determination (the sample set contains 188 entries). The AIC values are listed in Table 4.

**Table 4.** The values of AIC for different chain orders

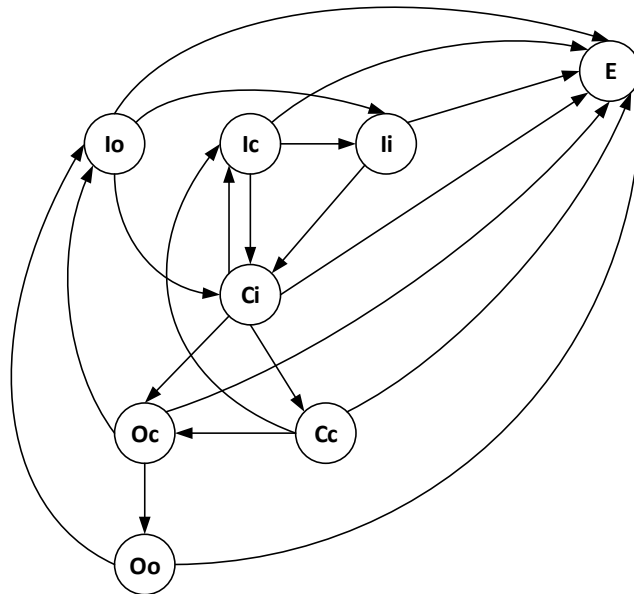
<b>The process order</b>	<b>AIC value</b>
1	223.7
2	203.5
3	352.7
4	1082.8

The initial graph  $\{C_i\}$  indicating the program components and control flow, as well as graph  $\{C'_i\}$  of the equivalent second order (see Table 4) Markov process are shown in Fig. 2 and 3 correspondingly. The notations in these figures are as follows: **I** correspond to *Input* state, **C** – to *Calculation*, **O** – to *Output*, and **E** to *Exit* state (see Table 2); indexes indicates the previous state in control flow history (thus state **O<sub>c</sub>** means that the current state *Output* has been reached from previous state *Calculation*).





**Fig.2.** Initial graph, representing the components and control flows of the software used for continuous time reliability model verification.

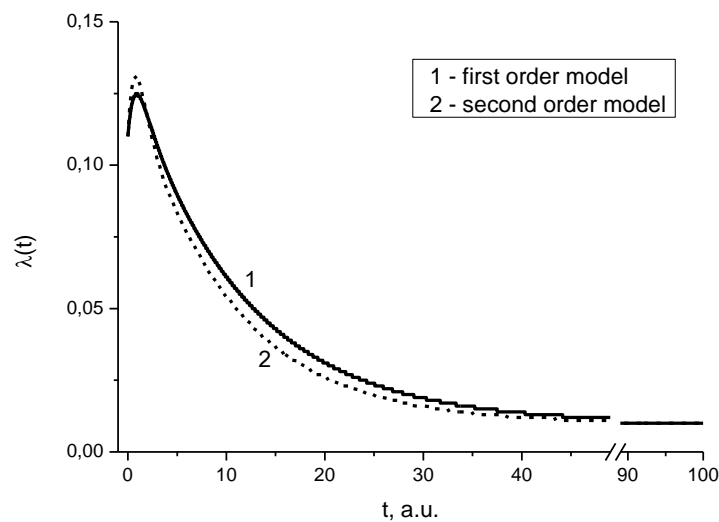


**Fig.3.** Graph representing an equivalent to the second order Markov chain process for software shown in Fig. 2.

Fig. 4 shows the time dependency of failure rate on tested software system (Fig. 2) obtained from continuous time first- and second-order Markov chains. It's worth noting that for continuous time model simulations time was counted as arbitrary time units (a.u).

As it is seen from this figure the first order model gives slightly (1–5%) reduced failure rate value for small values of time, while it gives significantly (20%) overestimated failure rate value for middle time values range, and when the value of  $t$  increases by more than 60 a.u. the difference between values of  $\lambda(t)$  obtained from

both models almost tends to be negligible. Reducing the difference of  $\lambda(t)$  values, obtained from the first and the second-order models, to zero at high times can be explained by decreasing of the  $\lambda(t)$  value itself and by the absence of differences in the software behaviour description by two models in this case. In the case of  $t \rightarrow \infty$  both models suggest that system is in the “Exit” state (see Fig. 2, 3) and, accordingly, its failure rate is limited by this component failure rate. Differences of  $\lambda(t)$  behaviour on the initial evolution stages of software system can be entirely explained by differences in the software system behaviour description by different models, where the first-order model ignores the interdependence of software component runs, and therefore the probabilities of components executing are different at given time  $t$  for both models. Thus, it could be argued that software reliability model based on the high order Markov chain describes the software system behaviour more adequately and determines its reliability measures more accurately.



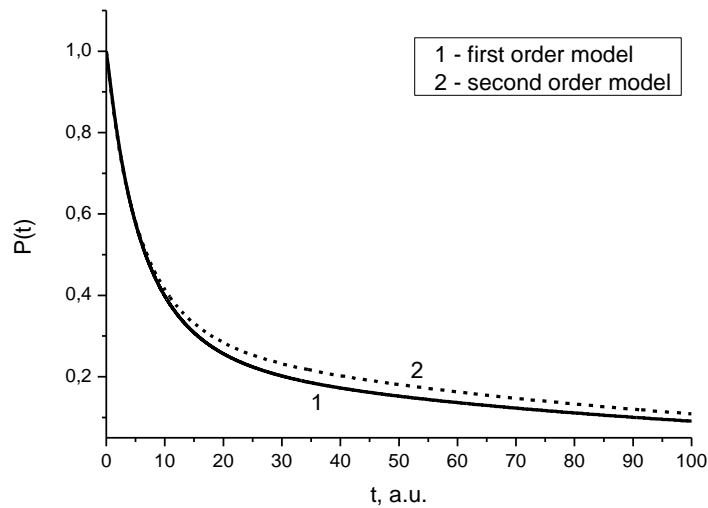
**Fig.4.** Time dependence of the tested software failure rate, obtained from the first (1) and second-order (2) models.

This conclusion is confirmed by the analysis of software system reliability function time dependence obtained using equation (7) as shown in Fig. 5. Note that the reliability function is calculated using Ukrainian national standards [22] and, as it was indicated in (7), represents the probability of failure free operation during the time interval  $(0, t]$ , but not exactly at the time  $t$ . So, it is evidently that the reliability function is time-decreasing one. As it can be seen from Fig. 5, the difference in the values  $P(t)$  increases up to  $\sim 20\%$  with time value increasing. This behavior is easily

understood if taking into account the interval estimation of reliability for the range  $(0, t]$  [15], and its deviation evidently increases with interval length increasing.

So, we can conclude that ignoring the interdependence of software components runs could result in increasing inaccuracy of software reliability measures estimation up to 20%.

Another measure of reliability, which was used to determine the developed high-order model effectiveness and adequacy, is the mean operating time to failure  $T_1$ . The value of mean operating time to failure calculated by the expression (8) is 20.9 time units for first-order model, while the value obtained from the second-order model is 23.3 time units. Obviously the difference between the first and the second-order models is 11.5%, which can be crucial in the reliability analysis of the complex technical systems.



**Fig.5.** The time dependence of the tested software reliability function, obtained from the first (1) and second-order (2) models.

Therefore, the usage of software reliability models based on the high-order Markov chains even with a small components number (in this case there was 4 component software, see Fig. 2) and complexity (in this case the optimal model order is 2, see Table 4) results in the increase of model adequacy and the software reliability assessment accuracy on 10–20%. The value of such growth of reliability estimation accuracy is especially important in the case of complex hardware and software systems in which the mistake of software component reliability estimation can repeatedly affect the accuracy of whole system reliability assessment. Using the high-order models for more complex software systems could result in more significant improvements of reliability assessment.

## 6 Conclusions

In this paper the software reliability models using discrete and continuous time high-order Markov chains, which enable consideration the dependencies among software components runs, have been developed. Software reliability assessment based on the high-order discrete time Markov chains allows to increase the software reliability measures accuracy up to 6 %, and high-order continuous time Markov chain – up to 10–20%. The advantage of high-order discrete time Markov chains model for software reliability analysis is small computing resources needed even for software with a lot of components, while the advantages of high-order continuous time Markov chains model are independence from the sampling step (in the case of discrete time model it may cause inaccurate transition probabilities calculation), and also automatic consideration of transitions  $p_{ii}$  that avoids unnecessary computations. Practical aspects of estimating input parameters for the models as well as studying the dependence of the software reliability on its characteristics (components number, the average number of variables in the component, components cohesion etc.) will be described elsewhere.

## References

1. Lyu, M. R.: Handbook of Software Reliability Engineering. McGraw-Hill, New York, U.S.A. and IEEE Computer Society Press, Los Alamitos, California, U.S.A. (1996)
2. Standard Glossary of Software Engineering Terminology, STD-729-1991 (1991)
3. Musa, J.D.: Validity of Execution Time Theory of Software Reliability. IEEE Trans. on Reliability 3, 199–205 (1979)
4. Goel, A. L., Okumoto K.: Time-Dependent Error Detection Rate Model for Software and other Performance Measures. IEEE Trans. on Reliability R-28, 206–211 (1979)
5. Xie, M.: Software Reliability Modelling. World Scientific, Singapore (1991)
6. Goševa-Popstojanova, K., Trivedi Kishor S.: Architecture-Based Approach to Reliability Assessment of Software Systems. Performance Evaluation, 45, 179–204 (2001)
7. Gokhale, S.S., Wong, W.E., Horgan, J.R., Trivedi Kishor, S.: An Analytical Approach to Architecture-Based Software Performance Reliability Prediction. Performance Evaluation 58(4), 13–22 (2004)
8. Takagi, T., Furukawa, Z., Yamasaki, T.: Accurate Usage Model Construction Using High-Order Markov Chains. In: Supplementary Proc. 17th Int. Symposium on Software Reliability Engineering, pp.1–2 (2006)
9. Burkhart, W., Fatiha, Z. (Eds.): Testing Software and Systems. Proc. 23rd IFIP WG 6.1 Int. Conf. LNCS, vol. 7019 (2011)
10. Yakovyna, V., Serdyuk, P., Nytrebych, O., Fedasyuk, D.: High-Order Markov Chains Usage in Software Reliability Analysis. Bulletin of Lviv Polytechnic National University 771, 209–213 (2013) (in Ukrainian)
11. Fedasyuk, D., Yakovyna, V., Serdyuk, P., Nytrebych O.: Variable State-Based Software Usage Model Based on its Variables. Econtechmod 3, 15–20 (2014)
12. Yakovyna, V., Masyukevych, V.: The Model for Software Reliability Estimation Using High-Order Continuous-Time Markov Chains. In: Proc.9th Int. Scientific and Technical Conference on Computer Science and Information Technologies, pp. 83–86 (2014)

13. Seniv, M., Yakovyna, V., Chabanyuk, Ya., Fedasyuk, D.: The Method of Reliability Prediction and Estimation Based on Model with Dynamic Index of Project Size. *Computing* 10, 97–107 (2011) (in Ukrainian)
14. Polovko A., Gurov, S.: *Fundamentals of the Reliability Theory. Practical work.* BHV-Peterburg (2006) (in Russian)
15. Markov Models, Hidden and Otherwise, <http://kochanski.org/gpk/teaching/0401Oxford/HMM.pdf>
16. Volochiy, B.Yu., Ozirkovskii, L.D., Kulyk, I.V.: Formalization of Discrete-Continuous Stochastic Systems Model Building using Erlang Phases Method. *Vidbir i Obrobka Informatsii* 36, 39–47 (2012) (in Ukrainian)
17. Burnham, P.: *Model Selection and Multimodel Inference: a Practical Information-Theoretic Approach.* Springer, Heidelberg (2002)
18. Liu, T.: Application of Markov Chains to Analyze and Predict the Time Series. *Modern Applied Science* 4(5), 162–166 (2010)
19. Akaike, H.: A New Look at the Statistical Model Identification. *IEEE Trans. Auto. Control* 19(6), 716–723 (1974)
20. Tong, H.: Determination of the Order of a Markov Chain by Akaike's Information Criterion. *J. Appl. Probability* 12, 488–497 (1975)
21. Schwarz, G.: Estimating the Dimension of a Model. *Annals of Statistics* 6(2), 461–464 (1978)
22. *Dependability of Technics. Terms and Definitions, State Standard of Ukraine DSTU-2860-94* (1996)