УДК 004.052

V. Yacovyna, P. Serdyuk, O. Nytrebych, D. Fedasyuk
National university "Lviv polytechnic"

Software department

THE USAGE OF HIGHER ORDER MARKOV CHAIN IN PROBLEMS OF MODELLING SOFTWARE RELIABILITY

Assumption of independent components execution in software reliability models built using architectural approach is a simplification of real software execution. In this paper Gokhale model with higher order Markov chains has been improved to appreciate software execution dependencies in it's reliability prediction.

Key words – software reliability, higher-order Markov chain, AIC criterion.

Introduction

In recent years, the complexity of computer programs that perform important and crucial functions is increasing rapidly, so the question of the software reliability is becoming more actual. Using adequate models of software reliability during the design and coding phases reduce the cost of testing phase. In this paper, according to [1] reliability means the probability of failure-free operation of a software system for a specified period of time in a specified environment. Today, there are many approaches and models which can predict or assess the software reliability. In common reliability models are divided into “blackbox” models and “whitebox” models [2] depending on the usage of information about software architecture.

Model is called “blackbox” when internal structure of software is unknown and conclusions can be obtained just via analysis of input/output data. In the last decade scientists pay more and more attention to “white-box” models (models which take into account information about the architecture of the software). In turn, architecture-based models are divided into Additive models, Path-based models and State-based models. It's obvious that “whitebox” models can describe software reliability in a more adequate way, because they evaluate internal structure of a software, i.e. it's architecture. That's why these models are also known as architecture-based software reliability models.

Today, a large number of articles devoted to the research a models of component -based approach using control flow graph which describes the architecture of a computer program [2, 3, 5, 6]. In this approach, software architecture can be modeled as Discrete Time Markov chain, Continuous Time Markov chain and Semi-Markov process [3]. Later, each of the models can be classified into absorbing (includes absorbing state - a state from which the system can’t get out) and irreducible (does not contain absorbing states). Besides this class of models can be further divided into composite and hierarchical models. The composite method combines the architecture of the software with the failure behavior into a composite model which is then solved to predict reliability of the application. The other possibility is the hierarchical approach, that is, to solve first the architectural model and then to superimpose the failure behavior on the solution of the architectural model in order to predict reliability. Models of this class mainly use the theory of first order Markov chains, suggesting that the implementation of software components is independent [3-5].

The appropriateness of high-order Markov chains usage for modeling such dependence between software components performance is shown in articles [1, 6]

Though, usage of high-order Markov chain requires solution of next problems:

- determination of the order for Markov chain;
- determination of transition probabilities between components;
- taking into consideration the conversion of stochastic to static model, in case of Markov chain order is equal or bigger than the training sequences;

- taking into consideration the quick enlargement of transition probability matrix with growth of Markov chain order.

This work is devoted to the practical aspects of using the higher order Markov chains to improve the increasing of software reliability prediction. The first section describes the using of AIC criterion for determining Markov chains, in the second chapter higher order Gochale modified model is demonstrated and the third section provides a new design software pattern for software implementation of higher-order Markov chains, which allows storing information for using such chains.

The determining of Markov chain order

We propose to use criterion AIC since it's not a hypothesis test and it doesn't use significance level. Besides, AIC gives consistent results and doesn't depend on models' calculation order.

In common case [7]:

$$AIC = 2k - 2\ln(L), \tag{1}$$

where k – number of independent parameters in the statistical model, L – model's maximum likelihood function value.

Consider we have software, which contains of S components (functional units that can be independently tested). Let's also assume that software architecture is being modeled by Markov process, which contains S states.

In terms of modeling Markov chains, Tong [8] proposed to use next risk function, based on AIC approach, which let to obtain the optimal order of Markov chain:

$$R(N) = -2\ln\lambda_{N,M} - 2(S^{M+1} - S^{M} - (S^{N+1} - S^{N})), \tag{2}$$

where M – the highest order of the model, N – order of the model being examined, $\lambda_{N,M}$ – The likelihood ratio under H_N to that under H_M (if $P_{ij\ldots kl}$ – is chain's transition probability, where suffix contains $N+1$ features, then hypothesis H_N: $P_{ij\ldots kl} = P_{j\ldots kl}$, $i = \overline{1,S}$; hypothesis H_M is defined similarly). The value of parameter N, which gives the minimum of function $R(N)$, is an optimal order for Markov chain.

When Markov chain order is determined, known mathematical apparatus of classical Markov processes can be used since any higher-order Markov process can be represented as first-order Markov chain.

Modified high-order software reliability Gokhale model

Gokhale model [9] is used quite often for software reliability assessment, in which software architecture is represented by discrete time Markov chains. This model is hierarchical, which means that initially parameters of architecture model are calculated, and then the behavior of failure components is taken into account for assessment software reliability.

According to Gokhale model the reliability of a computer system is calculated by the following equation:

$$R = \prod_{l=1}^{S} R_l, \tag{3}$$

here R_l – the reliability of each component, S – a number of software components. Applying high order Markov chains for this model (let N is an order of a model), the reliability of each component can be described as:

$$R_l = e^{\sum_i v_{ij..kl} \cdot t_{ij..kl} \int_0 \lambda_l(t)dt}. \tag{4}$$

In this equation $\lambda_l(t)$ – failure intensity of each component, $V_{ij..kl}$ – the expected number of visits a component *l* depending on being in previous *N* components, $t_{ij...kl}$ – time spent at component *l* depending on being in previous *N* components.

For obtaining the expected number of visits a component *i*, it should be calculated the total number of visits a component by formula [9]:

$$V_i = q_i + \sum_{j=1}^{S} V_j p_{ji}, i = \overline{1,S}, \quad (5)$$

where q_l – the initial state probability vector; p_{ij} – transition probability from component *i* to component *j*; V_i – expected number of visits of the respective components.

Then $V_{ij...kl}$ is obtained from:

$$V_{ij...kl} = V_i p_{ij...kl}, \quad (6)$$

where $p_{ij...kl}$ – transition probability to component *l* depending on being in previous *N* components.

The initial state probability vector, the transition probability matrix and time spent in the component can be obtained through program testing with previously implemented logging in components or using appropriate usage model [10].

To get the failure intensity of each component we can use "black box" models, based on the results of unit testing: Goel-Okumoto model, Musa, S-shaped or model with dynamic metric complexity of the project [11].

Using the numerical values of all model parameters, one can calculate the reliability of each component by the equation (4) and the value of the reliability of a computer system in general (3).

Markov chain dynamic representation model for reliability testing

Besides getting the software reliability model there are many open questions surrounding the assessment of software reliability, e.g. how to split software into components, to build the control flow graph and relative components runtime, etc.

There are miscellaneous assumptions on how to split software into components, define model parameters and apply them to real software in architecture-based approach for reliability assessment of software systems [2, 3].

In state-based models software component is a logically-independent module of the system, which performs a well-defined function [12]. However, in papers [3, 12] and others, software components are files, which isn't totally correct, since one file can contain a lot of logically-independent parts that interact with each other.

We propose to use software class as a software component, because it's a functionality unit, which can be independently tested using unit testing. Besides, it's possible to find the failure intensity, time spent in each class and transition probabilities to other classes [13].

We suggest usage of suffix tree to build control flow graph. This will enable us to build sequences of any length. We propose to use new software design pattern based on Composite pattern [14], pattern (this one is used to describe complex uniform structures) with modifications to represent transitions in graph. It's important that this pattern won't represent inexistent sequences unlike first-order Markov chains.

Modified class diagram for widely-used Composite design pattern, which can be used to get control flow graph, is shown in fig. 1 as an example.

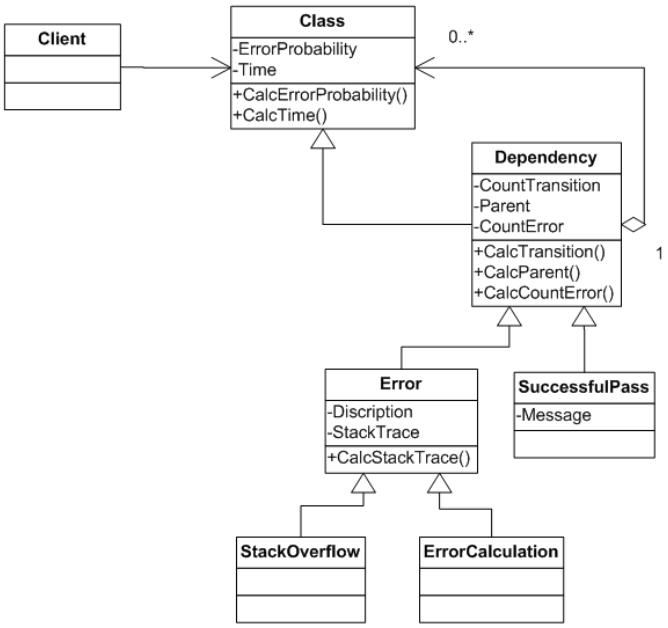

Fig 1. Modified composite design pattern which allows to represent suffix tree of Markov high-order chain of reliability testing.

Internal vertices will contain error counter and time, transitions counter will be stored in Dependency, leafs of suffix tree will represent errors or successful pass of a test. It's also important to store link to a parent to have ability to calculate probability. During execution of new tests only transition counter is changed, though general structure can change as well if new sequences are created. Thereby our pattern contains only required higher-order sequences, detailed information on transition probabilities, errors probabilities and errors themselves. The designed pattern has these advantages: spending less memory, dynamically changing the order and transformation for time-depended model.

Conclusions

In this work the usage of high-order Markov chains in the problem of modeling the software reliability in the case of the dependence components performance is analyzed. It is shown that the usage of AIC criterion optimizes and formalizes the process of determining the optimal order Markov chain in building software reliability model. A modified higher order Gochale model to assess the software reliability is proposed. For compact collecting information from high-order Markov chains we introduce new design pattern, which allows spending less memory, dynamically changing the order and transformation for time-depended model. Further research will be addressed to the practical confirmation of results and the practical application of the pattern and compared its with existing alternative approaches.

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. H. Pham "System Software Reliability" // Springer series in reliability engineering, Springer-Verlag London Limited (2006). 5. S. Krishnamurthy, A. Mathur "On the estimation of reliability of a software system using reliabilities of its components" // Proceedings of the Eighth International Symposium on Software Reliability Engineering (1997), 146–155. 6. T. Takagi, Z. Furukawa, T. Yamasaki "Accurate Usage Model Construction Using High-Order Markov Chains" // Supplementary Proceedings of 17th International Symposium on Software Reliability Engineering (2006), 1-2.

7. H. Akaike "A new look at the statistical model identification" // *IEEE Trans. Auto. Control. (1974), 716-723. 8. H. Tong, "Determination of the order of a Markov chain by Akaike's information criterion", Journal of applied probability 12 (1975), 488-497. 9. Gokhale S.S., Wong W.E., Horgan J.R., Kishor S. Trivedi "An analytical approach to architecture-based software performance reliability prediction" // Performance Evaluation 58, Issue 4 (2004), 391-412. 10. Heiko Koziolek "Operational Profiles for Software Reliability" // Dependability Engineering 2 (2005), 119-142. 11. Я.М. Чабанюк, В.С. Яковина, Д.В. Федасюк, М.М. Сенів, У.Т. Хімка "Побудова і дослідження моделі надійності програмного забезпечення з індексом величини проекту" // Інженерія програмного забезпечення 1 (2010), 24–29. 12. S.S. Gokhale, W.E. Wong, J.R. Horgan, S. Kishor, "An analytical approach to architecture-based software performance reliability prediction", Performance Evaluation 58 (4), 2004. 13. 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" // CADSM'2013, 382 (2013). 14. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес "Приемы объектно-ориентированного проектирования. Паттерны проектирования. " // СПб: Питер (2001), 368 с.*