MARKOV MODEL AND FUNCTIONAL SIZE WITH COSMIC-FFP

Manar Abu Talib
Concordia University
m_abutal@cse.concordia.ca

Alain Abran
École de Technologie Supérieure
aabran@ele.etsmtl.ca

Olga Ormandjieva
Concordia University
ormandj@cse.concordia.ca

***Abstract* - This research extends the architecture-based software reliability prediction model to the COSMIC-FFP context. This model is based on Markov chains and it is applicable prior to implementation with the ability to build reliability models much earlier at the requirement phase or based on the specifications for the design. In essence, each component of the system is modeled by a discrete time Markov chain. If this can be done, then a probabilistic analysis by Markov chains can be performed to evaluate the product reliability in the early phases of software development and to improve the reliability process for large software systems. This approach of applying a Markov model in the COSMIC-FFP context is illustrated with the railroad crossing case study.**

***Keywords:* Markov Model, Reliability Prediction, COSMIC-FFP, ISO 19761**

I. INTRODUCTION

Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment [1]. Software reliability engineering assumes usage-based statistical testing under the guidance of operational profiles that characterize usage patterns. Current software reliability models are applicable when the code is generated and is being tested.

Our research in this area concentrates on effective models and techniques which help evaluate product reliability in the early phases of software development, ultimately right from the requirements phase. The work reported here builds on our research results on the use of the COSMIC-FFP method for testing purposes by combining the functions measured by the COSMIC-FFP measurement procedure with a black box testing strategy [2], [3], [4].

COSMIC-FFP – ISO 19761 – is a functional size measurement method developed by the Common Software Measurement International Consortium (COSMIC) [5]. This measurement method focuses on the "user view" of functional requirements and is applicable throughout the development life cycle, right from the requirements phase to the implementation and maintenance phases. This measurement method has been designed to measure the functional size of management information systems, real-time software and multi-layer software systems. Since some of the software systems targeted by the COSMIC-FFP method are large-scale and inherently complex, feedback on this complexity could contribute to monitoring reliability throughout the software life cycle.

The research reported in this paper extends the architecture-based software reliability prediction model [6], based on Markov chains and applicable prior to implementation, to the COSMIC-FFP context. In essence, each component of the system is modeled by a discrete time Markov chain. The approach presented here of applying Markov model in the COSMIC-FFP context is illustrated with the railroad crossing case study.

The paper is organized as follows: Section II provides an overview of related work on the architecture-based reliability prediction model built from the theory of Markov chains. Section III presents the key elements of Markov model, and section IV the key elements of COSMIC-FFP. Section V discusses the mapping of concepts across the two fields, and, finally, section VI identifies research directions, including the use of both types of measures for the reliability prediction of software.

II. RELATED WORK

To assess reliability prior to implementation, it is important to understand the complex interactions among entities in software: this can be achieved by modeling the system (its objects, their interactions and the probabilities of the interactions) as a Markov system, and assessing the level of reliability through calculating the probabilities of their interactions. Such an approach yields results for both the time-dependent evolution of the system and the steady state of the system.

In [7], [8], a methodology was proposed for the uncertainty analysis of architecture-based software reliability models suitable for large, complex, component–based applications and applicable throughout the software life cycle. Within this methodology, two methods for uncertainty analysis have been developed: the method of moments and Monte Carlo simulation. In [8], the method of moments is used to quantify the uncertainty in software reliability due to uncertainty in component reliabilities. The expressions derived in [8] are valid for independent random variables and did not allow the uncertainty in software reliability to be studied due to uncertainty in the operational profile. Generalizing earlier research work on the method of moments, these authors then derived expressions for the mean and the variance of system reliability which consider both sources of uncertainty in software reliability (the way software is used, i.e. the operational profile) and the component's failure behavior (i.e. component reliabilities). This was illustrated through case studies in which the estimated values of the system reliability moments provide more information than the traditional point

estimate. Thus, these authors have a higher level of confidence in the reliability predictions for systems with a reliability having smaller variance. This information is especially useful in making predictions early in the life cycle, in keeping track of software evolution and in certifying the reliability of component-based systems.

The industrial applicability of Markov chains derived from state machine diagrams for reliability purposes is also worth mentioning, as described in [9], [10], [11]. For instance, the RELEX Markov [9], which provides fast, accurate reliability analyses for complex systems with common-cause failures, degradation, induced or dependent failures, multi-operational state components and other sequence-dependent events. Once a state transition diagram has been completed, the Markov engine incorporates optimized algorithms to perform calculations accurately and supports both transient and steady-state analysis results. In addition to calculating overall system results, the RELEX Markov also calculates parameters for each state.

The ability to take into account the measures of functionality early on, such as with COSMIC-FFP, makes it possible to consider the uncertainty in the operational profile (i.e. the uncertainty of environmental events) in addition to the uncertainty in component failure behavior, based on:

- Knowledge of the software architecture requirements (corresponds to the layers in COSMIC-FFP where objects behavior of software can be modeled with state diagrams within a layer, and then the Markov model is applied for these modules in one layer).
- Prediction of component reliability, a component is an object in software whose behavior is modeled in a state machine diagram.
- Probabilities of control transfer between components. These probabilities will be calculated as shown in [6], where the external events are considered first and each has equal probability if they are triggered from the same state, which is 1/n (where n is the total number of external events); the internal events are assigned the "left-over".

The reliability assessment method discussed in this work differs from previous reliability evaluation methods in the following ways:

- It is based on the architecture model of software and the state diagrams of software components.
- The prediction of reliability is derived from the steady state of the Markov system.

Moreover, this approach allows the reliability model to be applied at the design specification phase.

III. Markov Model

The discrete time Markov chain ([6], [12] and [13]) is a powerful mathematical tool for scientists and engineers analyzing and predicting the behaviors of a complex system. A Markov model analysis can yield a variety of useful performance measures describing the operation of the system, such as system reliability, availability, mean time to failure (MTTF), mean time between failures (MTBF), the probability of being in a given state at a given time, etc. There is interest in using Markov models for software reliability prediction purposes, since:

- Environmental laws are considered random and not controlled by system laws;
- Being in a particular state, a system may choose to execute any of the transitions available in that state in order to move to another state.

B. Markov Property & Markov System

A Markov process is a stochastic one which has two main characteristics:

1. It can take on a finite number of possible states, which we will index by the non-negative integers: 0, 1 ... and so on.
2. It has what is known as the "Markovian" property: the probability distribution of future states of the process depends only on the current state, and is conditionally independent of past states (the path of the process).

In other words, a Markov system can be in one of several mutually exclusive states, and can pass from one state to another according to fixed probabilities. For example, if a Markov system is in state *Si*, there is a fixed probability *pij* of it going into state *Sj* at the next time step. Therefore, the transition matrix is defined as matrix *P*, the *ij*-th entry of which is *pij*, and the entries in each row add up to 1.

C. Markov Model and State Machine Diagrams

A state-machine diagram, referred to as a state diagram in UML 2, is a UML behavioral diagram. It is used to model the dynamic behavior of individual objects and depict the various states that an object may be in and the transitions between those states. A state represents a stage in the behavior pattern of an object, and it is possible to have initial states and final states. A transition is a progression from one state to another and will be triggered by an event that is either internal or external to the object. See Figure 1 for an example of a state machine diagram that models the behavior of the object "Train" for the following railroad system case study [14] where more than one train can cross a gate simultaneously, through multiple parallel tracks. According to the train's destination, the train can independently choose the gate it will cross. Each gate is controlled by one controller which must be active all the time to close and open the gate for the railroad crossing. A train enters the crossing within an interval of time units after informing the controller that it is approaching. It also informs the controller that it is leaving the crossing within some time units of sending the approaching message. The controller, in response, commands the gate to close when it receives a message from the first train entering the crossing to make sure that no other train can cross the railroad at the same time. It also instructs the gate to reopen when it receives a message from the last train leaving the crossing.

The state machine in Figure 1 models the behavior of the "Train" at a point where it has five states: one initial state (the

black circle), “idle”, “toCross”, “cross”, “leave” and no final state since it is endless operation that is never stopped. It is to be noted that “Near” is a triggering event that makes the “Train” move from the “idle” state to the “toCross” state. Some of the transitions have conditions, such as (time units> entrance time > 0), which have to be satisfied in order for the object to move to other states. The general description for the following state machine diagram is that, as the train approaches a gate, it sends a “Near” message to the gate controller. Once the train leaves the gate, it sends an “Exit” message to the gate controller.

Which transition will be triggered from one state is the same as a random walk; based on this, the Markov model can be used to analyze the reliability of state machine software [6]. Therefore, the prediction of reliability is derived from the steady state of the Markov system. The mapping of the train object to a Markov system is shown in Figure 2, with a probability of 1 for each event, since there is only one event from each state. In the case where there are two events from one state, then each event will have a probability of ½. *P12* represents the transition probability that the event will be triggered, and the move is accordingly made from state *S1* to state *S2*.

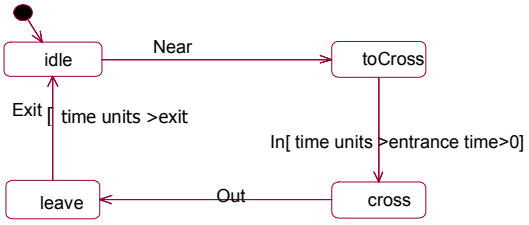

Figure 1 Train State Machine Diagram

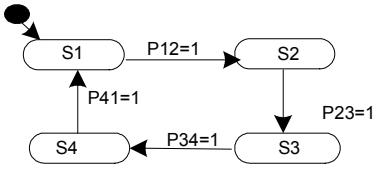

Figure 2 Train State Diagram with its Transition Probabilities *Pij*

Now, the transition matrix *P* (see Figure 3) can be built from this state machine diagram, and it is a matrix *P* whose *ij*-th entry is *Pij*. It is to be noted that the entries in each row add up to 1.

	S1	S2	S3	S4
S1	0	1	0	0
S2	0	0	1	0
S3	0	0	0	1
S4	1	0	0	0

Figure 3 Transition Matrix P for Train Object

The steady vector of the train object can then be calculated using the *P* matrix:

|[wxyz]| *P* = |[wxyz]| ➔ w = 0.25, x = 0.25, y = 0.25, z = 0.25.

Therefore, the steady vector is: [0.25, 0.25, 0.25, 0.25]

IV. COSMIC-FFP Measurement Method

A. COSMIC-FFP Overview

The functional size measurement method developed by the Common Software Measurement International Consortium (COSMIC) has now been adopted as an international standard (ISO 19761 [15]) and is referred to as the COSMIC-FFP method [5]. Its design was developed to address some of the major weaknesses of the earlier methods – like FPA [16], the design of which dates back almost 30 years when software was much smaller and much less varied.

In the measurement of software functional size using the COSMIC-FFP method, the software functional processes and their triggering events must be identified. In COSMIC-FFP, the unit of measurement is the data movement, which is a base functional component which moves one or more data attributes belonging to a single data group. Data movements can be of four types: Entry, Exit, Read or Write. The functional process is an elementary component of a set of user requirements triggered by one or more triggering events, either directly or indirectly, via an actor. The triggering event is an event occurring outside the boundary of the measured software and initiates one or more functional processes. The sub processes of each functional process are sequences of events, and comprise at least two data movement types: an Entry plus at least either an Exit or a Write. An Entry moves a data group, which is a set of data attributes, from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the boundary to the user requiring it. A Write moves a data group lying inside the functional process to persistent storage, and a Read moves a data group from persistent storage to the functional process. See Figure 4 for an illustration of the generic flow of data through software from a functional perspective.

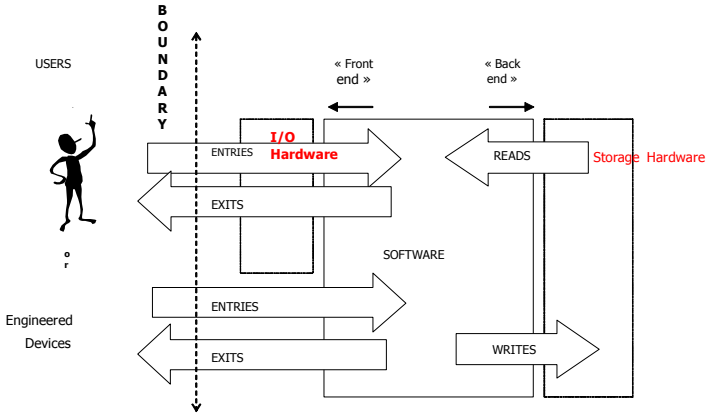

Figure 4 Generic Flow of Data through Software from a Functional Perspective [5]

B. COSMIC-FFP and Sequence Diagrams

A sequence diagram is a UML structural diagram that models the flow of logic within the system in a visual manner, enabling both the documentation and validation of the user's logic, and is commonly used for both analysis and design purposes. The sequence diagram is the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within the system. It consists of a group of instances (represented by lifelines or dashed lines) and the messages they exchange during the interaction (see Figures 5 and 6) that are the sequence diagrams derived from the railroad case study for functional size measurement purposes. Both have three objects, namely train, controller and gate, which interact by sending messages to each other. While measuring the functional size of software using COSMIC-FFP, the sequence diagrams are drawn to define the interactions between the software and its environment and within the software, as illustrated in Figure 4. In COSMIC-FFP, the environment is represented by the users interacting with the software, such users being humans, engineering devices or other software applications. Within the software, the interactions deal with the data read, or send the data to persistent storage. Going back to the railroad case study, the controller is the software that has a boundary where the trains interact with the controllers through sensors (many–to-many relationships) and the controllers communicate with the gates through actuators (one-to-one relationships). In the RUP context [17], the functional processes used in COSMIC-FFP can represent the set of scenarios for the software. For example, in the railroad system, the first sequence diagram (Figure 5) shows that, when a train arrives, a Near message is sent to the controller. The controller then instructs the gate to lower, and, in return, the gate goes down and the train enters the crossing. This process of allowing the train to cross the railroad is considered as a functional process, and is triggered by sending a Near message. Similarly, exiting the train (Figure 6) is a scenario containing a sequence of events between the train and the controller, and this scenario also contains a sequence of events within the system (controller in this case). Therefore, for each functional process, its sub processes and its triggering events are sequences of events (or data movements).

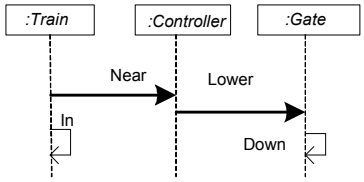

Figure 5 Train Enters Crossing Sequence Diagram

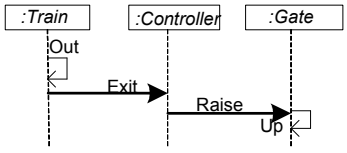

Figure 6 Train Leaves Crossing Sequence Diagram

V. Analysis of Linkages across Models

The functional size measurement method COSMIC-FFP can be linked to UML 2.0 state diagrams for modeling the behavior. This allows for probabilistic reliability modeling based on discrete Markov chains, since a Markov model is based on state diagram descriptions. The linkages between COSMIC-FFP and the Markov model can be analyzed, since the two have something in common, which are the UML diagrams.

The correspondence of COSMIC-FFP to UML state diagrams requires a mapping of COSMIC-FFP concepts (boundary, layer, functional process, triggering event, data group, movement and attributes, etc.) to state diagram notation. The reliability requirements for autonomic elements and systems have to be specified formally and mapped to system behavior so that the achievement of reliability can be monitored automatically. Analysis through Table 1 reveals that the same conceptual level is used for both COSMIC-FFP and UML 2 state machine diagrams; however, the terms used in the data movements of COSMIC-FFP and in the events of state machine diagrams have different labels. A summary of the terms used in both COSMIC-FFP and state machine diagrams that have similar meanings is presented in Table 1. For example, in COSMIC-FFP, data movements are classified in four categories: Entry, Exit, Read and Write. The term corresponding to the data movement and its categories that is used in state machine diagrams is "event", with two classifications: internal and external. In addition, data groups, which represent the set of data attributes in COSMIC-FFP, correspond to the term "objects" that is used in state machine diagrams. These diagrams explore the detailed transitions between states as the result of events (either external or internal) for only one object. Some additional expressiveness of the state machine diagrams could be taken into account. For instance, an external event can produce a set of internal events and this relationship (between internal and external events) probably can affect the software functional size and should be described in the sequence diagrams in terms of Entry and/or Exit data movements which may produce a set of Read and/or Write data movements. Another issue that can be carefully analyzed is the possible additional readings that may arise as a result of pre and/or post-conditions, where its operands can refer to other objects, associated to the events of a sequence or state diagram. That may affect the reliability prediction calculations based on Markov chains and its probabilities where conditional probabilities can be applied. Other terms used in both models, such as those interacting with the software, the software boundary and the set of user requirements, have the same labels.

From the mapping of concepts documented in Table 1, COSMIC-FFP and UML state machine diagrams have similar concepts. This motivated investigating the possibility of deriving state machine diagrams from COSMIC-FFP notations. It is to be noted that, while sequence diagrams have been used

in the COSMIC-FFP measurement method to explore the behaviors of one or more objects throughout a given period of time, the state machine diagrams for each object in COSMIC-FFP can be used to explore all their details.

Concepts	COSMIC-FFP (Data Movement) terms	State Machine Diagrams (Events) terms
Humans or things interacting with the software	Software users	Software users
Between the environment and the software	Software boundary	Software boundary
Set of User Requirements	Functional Process	Sequence of Events (Scenario)
Data which are part of the interaction	Data groups	Objects
External Input (From Environment)	Triggering event	External event
External Input (From Environment)	Entry data movement	External event
Output (to the environment)	Exit data movement	External event
Internal Input (Within Software)	Read data movement	Internal event
Internal Input (Within Software)	Write data movement	Internal event

Table 1 COSMIC-FFP & State Machine Diagrams

According to the COSMIC-FFP definitions given in its manual and the sequence diagrams that are drawn based on it, state machine diagrams can be derived from these sequence diagrams.COSMIC-FFP measurements can be mapped to UML 2.0 state diagrams using the technique proposed in [18] and illustrated with state machine diagrams from multiple interrelated scenarios (or sequence diagrams). A number of authors have discussed the way to transform a set of scenarios (or sequence diagrams) into state machine diagrams. However, the work proposed in [19] includes the steps and rules for deriving state machine diagrams from multiple scenarios with regard to the relationships between them. These rules are summarized as follows:

Step 1. Identifying and representing all single scenarios as sequence diagrams.

Step 2. Identifying and representing the relationships between all scenarios as dependency diagrams based on time dependencies between scenarios, their cause-effect dependencies and their generalization dependencies. The dependency diagram must have a single start point, which is the initial scenario, but it can have several end points.

Step 3. Synthesizing the state machines diagrams, based on the information acquired in the previous two steps.

Step 4. Refining the final state machines and approving the consistency between scenarios and state machines in order to make sure that the behavior of the final state machine diagrams reflects the information contained in the scenarios.

Now that the linkage between COSMIC-FFP and UML 2.0 state diagrams has been identified, the state machine diagrams can be derived accordingly. Going back to the railroad crossing case study, step 1 has already been performed in section IV (C), where the sequence diagrams are drawn for COSMIC-FFP purposes. Figure 7 shows the dependency diagram needed in step 2, i.e. the relationships between the scenarios (or sequence diagrams) and the order of execution.

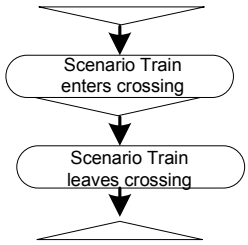

Figure 7 Dependency Diagram

One scenario is represented as a rounded rectangle, with connectors for the start point and end point. The initial scenario is "Scenario train enters crossing". At that point the train crosses the railroad, and the next scenario starts its execution, which is "Scenario train leaves crossing". This is simply a dependency diagram, where there are no alternative scenarios. Step 3 uses the information obtained in the previous steps to derive the corresponding state machine diagrams. It is to be noted that each sequence diagram shows the sequence of events (or data movements in the COSMIC-FFP context). Each event is a tuple: (*Oi, Oj, Mijk*), where *Oi* and *Oj* belong to the set of objects involved in the software and *Mijk* is the message that is exchanged between them. Therefore, the sequence diagram in Figure 5 has the following set of tuples = {(train, controller, Near), (train, train, In), (controller, gate, Lower), (gate, gate, Down)} and the sequence diagram in Figure 6 has the following set of tuples = {(train, train, Out), (train, controller, Exit), (controller, gate, Raise), (gate, gate, Up)}. There are three objects involved in each scenario, and therefore we can synthesize three state machine diagrams (one for each object).

For each object, one initial state machine diagram can be created for each scenario, and the final state machine diagram can then be synthesized from all the state machine diagrams, based on the information in the dependency diagrams. The state machine diagram for the train object in Figure 1 is obtained from two initial state machine diagrams shown in Figures in 8 and 9.

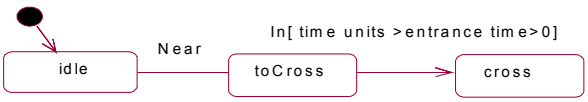

Figure 8 Initial State Machine Diagram from Figure 5

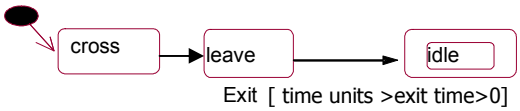

Figure 9 Initial State Machine Diagram from Figure 6

Similarly, state machine diagrams for the objects controller and gate are created as shown in Figures 10 and 11.

VI. DISCUSSION AND NEXT STEPS

In this paper, the candidate linkages between the Markov models and the functional size measurement method COSMIC-

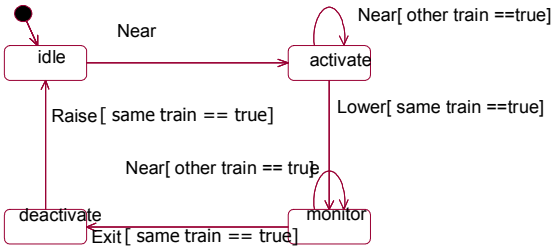

Figure 10 Controller State Machine Diagram

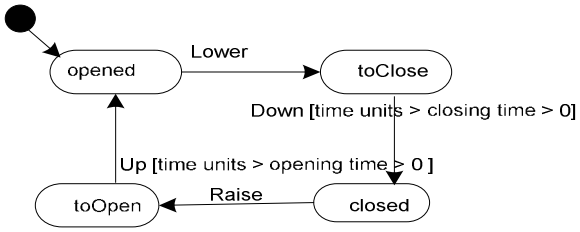

Figure 11 Gate State Machine Diagram

FFP are investigated for the reliability prediction of software based on Markov concepts in a COSMIC-FFP context. This is achieved by first synthesizing the corresponding state machine diagrams from the COSMIC-FFP sequence diagrams. Second, a Markov system is formalized by using the derived COSMIC-FFP state machine diagrams. Third, the steady state distribution vector for the corresponding Markov system is calculated.

Research in progress is looking into the reliability prediction calculations. The reliability prediction for a system composed of *n* objects can be defined as the level of certainty quantified by a level of uncertainty in a Markov system corresponding to an object, and a level of uncertainty of a Markov system corresponding to a subsystem. Suggestions for future works include investigating the use of predictions to compare alternative systems designs, and gathering data from empirical studies to assess the effectiveness of the reliability model and the degree of confidence of the predicted values. Moreover, a comparison (if available) with results obtained using alternative methodologies to support the validity of the application of our proposed methodology. Further work is progressing on the formalization of COSMIC-FFP in the context of AS-TRM (Autonomic Systems Timed Reactive Model), a language for the formal design of autonomic reactive systems. AS-TRM is based on the notion of extended state machines; this will allow the application of the rules of COSMIC-FFP functional size measurement to AS-TRM specifications, making it possible to analyze several case studies for validation purposes.

REFERENCES

1. Institute of Electrical and Electronics Engineers, *ANSI/IEEE Standard Glossary of Software Terminology*, IEEE Std. 729-1992, 1991.
2. Abu Talib, M., Ormandjieva, O., Abran, A. and Buglione, L., *Scenario-based Black-Box Testing in COSMIC-FFP*, 2nd Software Measurement European Forum, Rome (Italy), 16-18 March 2005, pp. 173- 182.
3. Abu Talib, M., Ormandjeva, O., Abran, A., Khelifi, A. and Bublione, L., *Scenario-based Black-Box Testing in COSMIC-FFP: A Case Study*, accepted in Software Quality (ASQ) Journal, to be published in 2006
4. Abran, A., Ormandjieva, O. and Abu Talib, M., *Functional Size and Information Theory-Based Functional Complexity Measures: Exploratory study of related concepts using COSMIC-FFP measurement method as a case study*, 14th International Workshop of Software Measurement (IWSM-MetriKon 2004), Shaker-Verlag, Konigs Wusterhausen, Germany, 2004, pp. 457-471.
5. Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D. and Symons, C., *COSMIC FFP – Measurement Manual (COSMIC implementation guide to ISO/IEC 19761:2003)*, École de technologie supérieure - Université du Québec, Montréal. 2003..
6. Ormandjieva, O., *Deriving New Measurement for Real Time Reactive Systems* Ph.D. dissertation, Department of Computer Science & Software Engineering, Concordia University, Montreal, Canada, 2002.
7. Go˘seva-Popstojanova, K. and Kamavaram, S., *Software Reliability Estimation under Uncertainty: Generalization of the Method of Moments*, Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE'04), 2004.
8. Go˘seva–Popstojanova, K. and Kamavaram, S., *Assessing Uncertainty in Reliability of Component-Based Software Systems*, 14th Int'l Symp. Software Reliability Engineering, Nov. 2003, pp. 307-320.
9. Relex software, URL: http://www.relexsoftware.com.
10. Item software, URL: http://www.itemsoft.com.
11. ISO graph, URL: http://www.isograph-software.com/index.htm.
12. Strook, D. W., *An Introduction to Markov Processes*, Springer-Verlag, Berlin, Heidelberg, 2005.
13. Trvedi, A.K., *Computer Software Reliability: Many-State Markov Modeling Techniques*, Ph.D. dissertation, Polytechnic Institute of Brooklyn, June, 1975.
14. Rajeev Alur, Lecture Notes in Computer Science, Springer-Verlag GmbH, ISSN: 0302-9743, Volume 1633 / 1999, Computer Aided Verification: 11th International Conference, CAV'99. Trento, Italy, July 1999. Proceedings, pp. 8 - 22.
15. ISO/IEC 19761. Software Engineering - *COSMIC-FFP - A functional size measurement method.* International Organization for Standardization - ISO, Geneva, 2003.
16. Albrecht, A.J. and Gaffney, J.E., Software Function, *Source Lines of Code, and Development Effort Prediction: A Software Science Validation*. IEEE Trans. Software Eng. vol. SE-9, no.6, pp. 639-648, Nov. 1983.
17. Kruchten, P., *The Rational Unified Process: An Introduction*, Addison-Wesley, 2003.
18. Vasilache, S. and Tanaka, Jiro, *Synthesis of state machines from multiple interrelated scenarios using dependency diagrams*. Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, Florida, USA, July 18-21, 2004, pp. 49-54.