SIMULATION AND RELIABILITY ANALYSIS OF CONTROL SYSTEMS WITH MULTIVERSION SOFTWARE

V.S. Kharchenko[1], V.V. Sklyar[2]

*[1]National Aerospace University named after N.E. Zhukovsky
"Kharkiv Aviation Institute",
[2]Kharkiv Military University, Ukraine*

Abstract: The solution of the problem of multiversion control computer-based systems (MVSs) simulation and reliability evaluation is offered by Monte-Carlo method. This method is applied for simulation of: software defects and defective versions; input data for different distribution laws; hardware and majority subsystem failures. Detailed notation of the modeling system and the simulation technique are given. The results of the simulation and reliability evaluation of the majority software versions and adaptive MVS are analyzed. *Copyright © 2002 IFAC*

Keywords: adaptation, digital systems, multiversion software, reliability analysis, simulation.

1. INTRODUCTION

The reliability and safety of computer control systems substantially depend upon software reliability. Over the past forty years the proportion of software defects among failure reasons has increased from 10-15 to 30-60 % (Lyu, 1996; Kharchenko, 1996; Lapri, 1998). First of all, it is concerned with computer control real-time systems for critical applications. Such systems are produced by unique designs and are of high cost. Moreover, the designing and testing technologies don't guarantee the elimination of all software defects. The decision of hardware and software defect tolerance problem is possible by application of a multiversity. The multiversion principle is realized in computer systems used in aviation, cosmonautics, chemical industry, railway transport, atomic power stations. The multiversion system (MVS) reliability, as any computer system reliability, is evaluated in view of their architecture and reliability of two components: software and hardware. If for determination of hardware reliability measures (RMs) the efficient mathematical apparatus is based on using combination-probability methods, then it is difficult for software to determine the laws of random distribution. It this case it is advantageous to use Monte-Carlo method.

The aim of the paper is development of the technique of simulation and reliability analysis of MVSs and multiversion software. The elements of this technique are the algorithm of software simulation, the approach to estimating of MVS reliability and the developed reliability model of the adaptive multiversion majority systems. In this paper it is proposed for MVS reliability estimating to combine two methods: combination-probability method for hardware reliability estimation and Monte-Carlo method for software reliability estimation and system as a whole. The subject of investigation is restricted to MVSs for unmanned computer control real-time systems used, for example, in aerospace complexes (Schneidewind, 1997).

2. GENERAL ALGORITHM OF SOFTWARE RELIABILITY ESTIMATION BY MONTE-CARLO METHOD

The process of software simulation for estimating its reliability consists of three stages: development of software model with defects; selection of distribution law and generation of software input data; estimation of software RMs. The general algorithm of software operation process is shown in Fig. 1. At the first stage for simulation of the process of defect

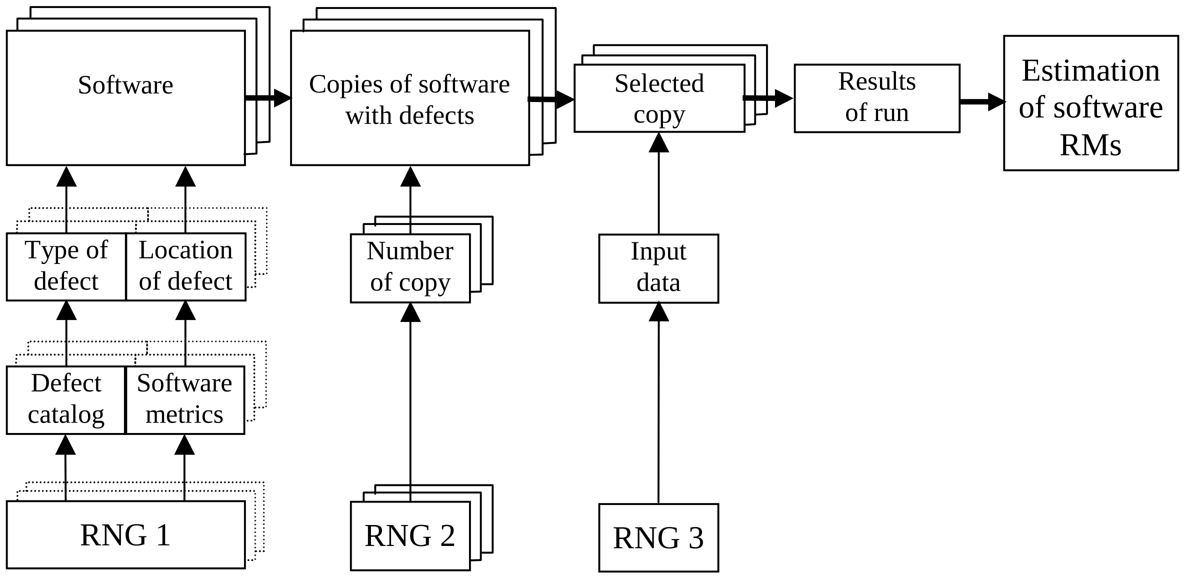

Figure 1. General algorithm of software operation process

introduction into the program by Monte-Carlo method, the types of defects and their location in program text are generated; for this purpose the random-number generator (RNG1) is used. As a result, non-identical software copies with defects are obtained. They constitute the database of software defect copies. The domains of values and the laws of random distribution for all input values being processed by the program are determined at the second stage of the simulation.

RNG2 must reproduce the serial number of software copy from defect copy database, and RNG3 – the sets of input data. The results of software operation are fixed and on the basis of these data the software RMs are calculated. Each simulation stage may be a one-version or multiversion one.

3. DEVELOPMENT OF SOFTWARE MODEL WITH DEFECTS

At the first stage of modeling process defects are introduced into initial software text. For this purpose RNG1, defects catalog and software metrics set are used (Fig. 1). Defects catalog includes the list of numbered defect types, the most characteristic for software being investigated. Metric is the numbers of software (modules, lines, operators, operands), into which the program is decomposed according to certain rules (Halstead, 1977; Lyu, 1996). In general case there can be used some different metrics for a single program. The type of defect (number from defect catalog) and its location (number of metric element) are selected by Monte-Carlo method.

For this, as a rule, the uniform law of random distribution is modeled. In general case the random distribution law can be modified. Then defect varieties in software can be shifted to some type of defects and their location can be grouped around certain position in program text. The number of defects in software is calculated with the help of apriority model, for example, Halstead model (Halstead, 1977). Their estimation then can be corrected in view of testing intensity. The number of software defects computed according to some model is usually small. If for simulation modeling only one copy of software with defects is analyzed, this can yield sufficient shift of estimations. Therefore, in a given methodology it is proposed to replicate a great number of software defect copies, thus, creating a database for defects. As more there are such software copies, as more accurate some “mean” software defects can be modeled and as more precise software RMs can be evaluated. At the same time manipulating with defect catalog contents and metrics set and also RNG1 parameters, it is possible to maximize an adequacy of defect copy database to designer’s peculiarities and design specificity.

4. GENERATION OF INPUT DATA SETS

It is proposed to model input data sets by Monte-Carlo method. Proceeding from technical task and operation conditions, for each input magnitude X_i, $i =$ $= 1,\dots,n$ the domain of possible values can be defined, even if approximately, i.e. the interval $X_i \in [X_{ib};X_{if}]$. Besides, proceeding from the same conditions for each magnitude X_i a distribution law must be given. In general case it can be a uniform random distribution law. Then RNG3 form n input data from sets X at given intervals according to selected distribution laws. The software testing is realized at these input data sets. Besides, RNG2 must specify $(n + 1)$-th random value, serving for determination of defect copy number from database, which must be called up for execution at particular testing stage (Fig. 1).

The simulated number of software defects can be corrected depending on life cycle phase. To do so, it is necessary to increase the number of module numbers generated by RNG2. Thus, if the number of a defect copy generated by RNG2 exceeds the number of copies in the database, then initial software is executed. If the number of defect modules remains constant, then the difference between a maximum value generated by RNG2 and the number of defect modules is increased. For uniform distribution the number of calls of defect modules is decreased and in the end reliability measures of software are increased. At different stages of software life cycle this process can be simulated by giving the coefficients of software debugging. It should be noted, that when the generated input data set is processed by initial software, the defects skipped in testing can be detected. Thus, the methodology allows to improve the quality of software debugging process and to refine software reliability estimation.

5. COMPUTATION OF SOFTWARE RELIABILITY MEASURES

For justification of calculated formulas it is necessary to form a series of theoretical propositions. Software operation of computer systems consists in a processing data set $\mathbf{X} = \{X_1,...,X_i,...,X_n\}$, that arrive at computer system input. By the term “run” software (single execution) it will be understand an event consisting of arrival at program input of arbitrary input data set $\mathbf{X}$, its processing by the program and producing desired results.

In custom-definition of requirements to the program it is supposed, that software will execute some function $\varphi(\mathbf{X})$. However, due to different inaccuracies of technological cycle of software development, the real software implements some function $\varphi^1(\mathbf{X})$ different from the given one $\varphi(\mathbf{X})$.

Reasons causing such inaccuracy are explained by complex conversions of information during software development; these conversions are impossible without some distortions. The analysis of distortion causes and mechanisms of their introduction is the subject of separate investigation. In most cases the difference in operation results between real and ideal software lies in the tolerances, i.e.

$$\Delta(\mathbf{X}) = \varphi(\mathbf{X}) - \varphi^1(\mathbf{X}) \le \Delta_{tol} \; . \qquad (1)$$

However, some cases are possible when difference of results exceeds a tolerable one, i.e.

$$\Delta(\mathbf{X}) = \varphi(\mathbf{X}) - \varphi^1(\mathbf{X}) > \Delta_{tol} \; . \qquad (2)$$

The event consisting of the fact that software operation results are differed from required results by magnitude exceeding tolerable deviation is fixed as fault of software. The term “fault” but not “failure” is used because unmanned control systems are considered; their operation cycle consists of series of computing cycles; at each cycle identical computations are performed for different input data. Hence, if $\Delta(\mathbf{X}) > \Delta_{tol}$ at i-th step (cycle) of computations, it is quite probably that at $(i + 1)$-th cycle we will have $\Delta(\mathbf{X}) \le \Delta_{to}l$ and normal operation of software, and system on the whole will be restored. It should be noted, that $\Delta(\mathbf{X})$ characterizes not only quantitative aspect of computing process, i.e. the accuracy of software operation or computation error. The magnitude $\Delta(\mathbf{X})$ characterizes and qualitative aspect of computing process, i.e. determines limitations on computing cycle time, absence of unusual situations, post-fault reset time etc. In unmanned systems software defects are not eliminated, hence the number of defect remains constant during the whole of operation stage. In turn this results that at fixed laws of input magnitude distribution the rate of defect display remains constant. Therefore, the mean time before software fault is distributed exponentially:

$$P(t) = \exp\{-\lambda t\} \; , \qquad (3)$$

where $P(t)$ – probability of non-fault of software over time t (PNF);
λ – fault rate of software (constant value).

To use this formula it is necessary to define the fault rate. The operation results of software model with defects are compared with required ones in the course of testing, and software faults are fixed. Then the probability of software single non-fault operation can be defined by the formula

$$P_1 = 1 - \frac{N_{faults}}{N_{runs}} \; , \qquad (4)$$

where N_{faults} – the number of fault fixed during testing;
N_{runs} – total number of software runs.

In unmanned systems software defects causing faults are not eliminated in the process of their functioning. Therefore at any serial number of computing step the magnitude P_1 will be the same. Fault rate of software in [1/step] can be defined by the formula

$$\lambda^*[1/\text{step}] = -\ln P_1 \; . \qquad (5)$$

Dependence of software PNF on step serial number (cycle, run) is defined by the formula

$$P(N) = \exp\{-\lambda^* N\} \; , \qquad (6)$$

where N – serial number of computing step of control system.

For the change from step serial number to time it is necessary to take account of the computing step duration T: λ[1/hour] = λ*[1/step] / T[hour]. Then

$$P(t) = \exp\{-\lambda NT\}\ . \qquad (7)$$

The example of software reliability assessment are illustrated by plots of version PNF change in time at uniform and normal laws of input data distribution (Fig. 2).

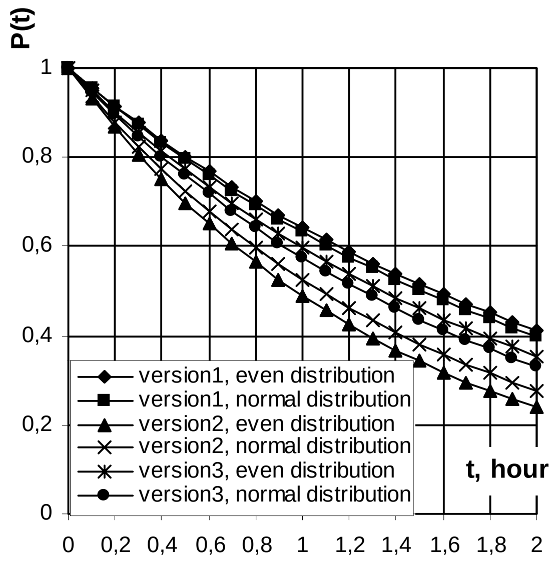

Fig. 2. Plots of software version PNF change in time

6. ESTIMATION OF MULTIVERSION SYSTEMS RELIABILITY

To estimate MVS reliability as a collection of interrelated elements is necessary take into account both software and hardware reliability. In MVS hardware as in any computer system two constituent parts can be distinguished: 1) primary hardware (denote it *HW*1, it includes various storage devices where instruction sequences are stored *HW*1 component volume is utterly determined by software volume); 2) computing equipment implementing information processing, signal conversion etc. (it is secondary hardware *HW*2, it includes processors, arithmetic-logic devices, data converters etc).

System failure or fault can be caused by physical defects of system elements and design defects. Physical defects are characteristic of *HW*1 and *HW*2 components and design defects are inherent in software, basically. Consequently, for objective estimation of computer system reliability it is necessary to take account of software component reliability (denote it *SW*). Thus, the system can be represented by three components

$$S = \{SW, HW1, HW2\}\ . \qquad (8)$$

The more developed the software is more capacious are the *SW* and *HW*1 components and the less is the *HW*2 component volume and vice versa. If it is supposed that *SW*, *HW*1 and *HW*2 failures are independent then the probability of non-failure of system will be equal to PNF product of corresponding components

$$P_S = P_{SW}(t)P_{HW1}(t)P_{HW2}(t)\ . \qquad (9)$$

As a rule the exponential distribution of time before failure is used for the computation of the hardware probability of non-failure:

$$\begin{cases} P_{HW1}(t) = \exp\{-\lambda_{HW1}t\}\,; \\ P_{HW2}(t) = \exp\{-\lambda_{HW2}t\}. \end{cases} \qquad (10)$$

As was shown above, for software of unmanned systems it is also necessary to apply the exponential law of time distribution before fault. It is a custom to divide software into applied or functional (SW^{Fi}, i = 1,…,n) and system (SW^{S}) software. In its turn, applied software executes a number of functions, which are determined by MVS specificity. Besides, the functions of applied software can be reduced to such a set Fi, i = 1,…,n, that non-execution of any set function will result in MVS failure. Primary hardware component can be divided into the following parts depending on software: hardware for storing system software ($HW1^{S}$) and hardware for storing applied software ($HW1^{Fi}$, i = 1,…,n). ($HW1^{Fi}$, i = 1,…,n). Proceeding from the above-stated, the structural model of one-channel computer system can be represented in the form shown in Fig. 3. The probability of non-failure of the system, shown in Fig. 3 is defined by the formula:

$$P_S(t) = P_{SW^S}(t)\prod_{i=1}^{n}\left[P_{SW^{Fi}}(t)\right]P_{HW1^S}(t)\times \\ \times\prod_{i=1}^{n}\left[P_{HW1^{Fi}}(t)\right]P_{HW2}(t) \qquad (11)$$

Using an assumption that all functional modules of software SW^{Fi}, i = 1,…,n are approximately equal in volume and complexity, the equation (11) will take the following form:

$$P_S(t) = P_{SW^S}(t)[P_{SW^{Fi}}(t)]^n\ P_{HW1^S}(t)\times \\ \times[P_{HW1^{Fi}}(t)]^n P_{HW2}(t)\ . \qquad (12)$$

When deriving formulas for MVSs it is necessary to take into account the following fundamentally important points.

1. Application of three-version majority architectures assumes availability of majority element (ME) in the system. As system the ME can be decomposed into the following components: SW^{M},

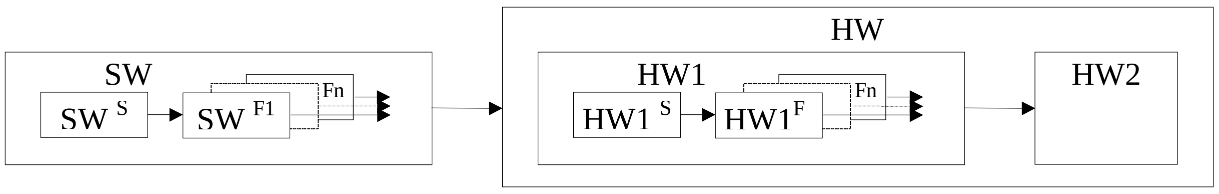

Fig. 3. Model of one-channel computer system

$HW1^M$, $HW2^M$. Probability of non-failure of these components must be taken into account in estimating MVS probability of non-failure.

2. Besides, it should be noted, that the portion of defects is common for all three versions in MVS. Such defects are called absolute ones, and different defects for each of the versions are called the relative ones. Presence of absolute defects may be caused by two reasons: a) occasional coincidence of programming errors in different versions; b) specification errors.

The first component of absolute defects can be estimated by Monte-Carlo method in the course of determining version defect rate. The second component is usually given by coefficient K_{SWabs} of the ratio of absolute defect display rate of multiversion software to software relative defect display rate. By different estimations (Lyu, 1996; Kharchenko, 1996) this coefficient varies within the limits of 0.1-0.3.

Thus, multiversion software fault rate has constant component, attributed to absolute defects, and variable component, attributed to version relative defects. The events consisting of software absolute and relative defect display are considered independent. These events are estimated by probabilities P_{SWabs} and P_{SWrel} and rates λ_{SWabs} and λ_{SWrel}, respectively. Then the probability of software non-fault operation

$$\begin{cases} P_{SW_{N-version}} = P_{SW_{abs}} P_{SW_{rel}}\ ; \\ P_{SW_{abs}} = \exp\{-\lambda_{SW_{abs}} t\} = \exp\{-K_{SW_{abs}} \lambda_{SW_{rel}} t\}\,; \quad (13) \\ P_{SW_{rel}} = 3\exp\{-2\lambda_{SW_{rel}} t\} - 2\exp\{-3\lambda_{SW_{rel}} t\}\,. \end{cases}$$

Finally, to define MVS one has to take into account the adaptation algorithm and the reliability of hardware and software checking as well as the diagnostic device (D). It should be noted, that the Monte-Carlo method can be used for modeling not only software faults, but for modeling hardware failures. To do so, it is necessary to introduce into the modeling scheme (Fig. 1) one more RNG and define time distribution laws before failure for software components.
Let the adaptation algorithm be used in MVS, under which MVS is reconfigured on one version or one hardware channel at failure of two of three software versions (two of three hardware channels). For all this it is unimportant, how many hardware channels failed before this event. As an example, the probability of non-failure of such MVS is equal:

$$P_S(t) = \{[3P^2_{SW^{F_i}_{rel}}(t)^2 - 2P^3_{SW^{F_i}_{rel}}(t) + 3P_{SW^{F_i}_{rel}}(t) \times$$
$$\times [1 - P_{SW^{F_i}_{rel}}(t)]^2 D]^n [P^n_{HW1^{F\,i}}(t) P_{HW1^S}(t) P_{HW2}(t)]^3 +$$
$$+ 3[1 - P^n_{HW1^{F\,i}}(t) P_{HW1^S}(t) P_{HW2}(t)] \times [P^n_{HW1^{F\,i}}(t) \times$$
$$\times P_{HW1^S}(t) P_{HW2}(t)]^2 \times [3P^2_{SW^{F_i}_{rel}}(t) - 2P^3_{SW^{F_i}_{rel}}(t) +$$
$$+ 3P_{SW^{F_i}_{rel}}(t)[1 - P_{SW^{F_i}_{rel}}(t)]^2 D]^{\,n} 3[1 - P^n_{HW1^{F\,i}}(t) \times$$
$$\times P_{HW1^S}(t) P_{HW2}(t)]^2 D P^n_{HW1^{F\,i}}(t) P_{HW1^S}(t) P_{HW2}(t) \times$$
$$\times [3P^2_{SW^{F_i}_{rel}}(t) - 2P^3_{SW^{F_i}_{rel}}(t) + 3P_{SW^{F_i}_{rel}}(t)[1 - P_{SW^{F_i}_{rel}}(t)]^2 \times$$
$$\times D]^{\,n}\} P_{SW^S}(t) [P_{SW^{F_i}_{abs}}(t)]^n P_{SW^M}(t) P_{HW1^M}(t) P_{HW2^M}\ .$$
(14)

For calculation according to formula (14) it is necessary to define the portion of software absolute and relative defects. Probability of non-failure plots of adaptive MVS for various values of absolute defect coefficient K_{SWabs} are shown in Fig. 4 (λ_{SWS} = λ_{SWFi} = 10^{-5} 1/h; n = 5; λ_{HW1Fi} = λ_{HW1S} = 10^{-7} 1/h; λ_{HW2} = 10^{-6} 1/h; λ_{SWM} =10^{-8} = λ_{HW1M} = λ_{HW2M} = 10^{-8} 1/h; D = 0,9).

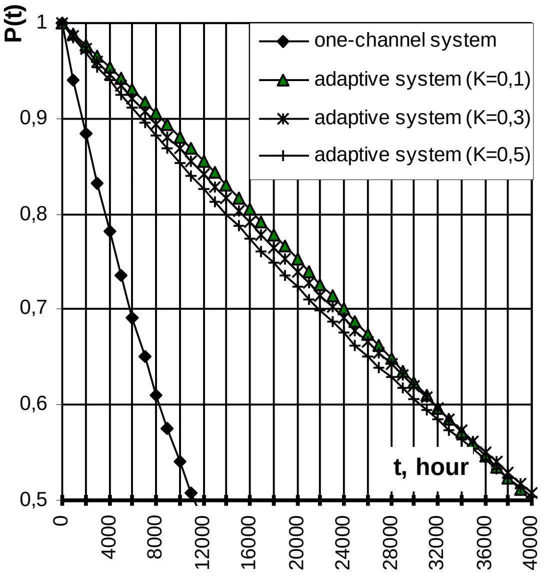

Fig. 4. Probability of non-failure plots of adaptive MVS

7. MULTIVERSION TECHNOLOGY OF SOFTWARE DEVELOPMENT FOR RELIABILITY ASSESSMENT

To decrease risks of erroneous reliability calculation connected to software faults a method is proposed which was tested in assessment, engineering and evaluation of aerospace, nuclear power plants safety systems and other critical applications. This method is based on multiversion technology of critical real-time software development (Lyu, 1996; Kharchenko, 1999).

Multiversion mode can be introduced at various stages (specification, mathematical model development, software design, coding, testing, verification, etc.) and by various methods development. Issuing from this a set of multiversion technologies is formed which is corroborated by multilevel graph in general case.

The analysis showed that an increase in cost of development of several versions may be compensated for by cost reduction due to shorter testing time sufficient for an acceptable residual level of software faults. Such a phenomenon is explained by a higher intensity of fault finding during parallel testing of several versions.

For the reliability assessment problem under consideration multiversion mode is introduced at the stage of formalization by selecting various types of mathematical models and at the stage of software design by application of independent program designers, various programming languages, code generators and debugging tools. The developed program versions were assessed using Halstead metrics, then qualified according to test results. In case of identity of version reliability final characteristics a program with the least value of defects number or another parameter was chosen as a basic (operational) one (Kharchenko *et al.,* 2000).

The procedure of choice among multiversion technologies is hard for formalization due to a large correlation of costs and time of realization of various lifecycle stages and attained software reliability level. The choice of technologies can be based upon expression (Kharchenko, 1996) interconnecting the probability of software no-fault operation after design and testing and total costs of performance of those two stages C_{DT}:

$$\begin{cases} P_{DT} = 1-(1-P_D)\beta^{\alpha}\ ; \\ \alpha = -\gamma(C_{DT} - e_N C_D)\ , \end{cases} \qquad (16)$$

where P_D – probability of software no-fault operation after design;
C_D – cost of unit version design;
β – testing process coefficient;
γ – coefficient depending upon complexity of program and number of versions.

8. CONCLUSION

The expansion of real-time MVS field of application calls for requirements to reliability estimation of such systems. To estimate reliability of complex objects, such as MVS, it is insufficiently to use one method. Therefore, for MVS reliability estimation joint application of combination–probability method (to estimate hardware component) and Monte-Carlo method (to estimate software component) is proposed.

The technology of analysis and choice of architectures of MVSs is developed on the basis of use of the different methods of reliability estimating. It is one of the variants of multiversion technologies (Kharchenko, 1996).

The proposed technique of simulation and reliability estimation of MVSs is the essential element of this technology. The technology is used for designing, testing and review of MVSs. This technology is “an open technology” for expansion of the field of Monte-Carlo method application for estimating reliability of hardware and majority and reconfiguration means. Such necessity may take place when very large-scale integrated circuits are used and their laws of failure distribution require refinement.

REFERENCES

Halstead, M. (1977). *Elements of Software Science.* Elsevier North-Holland, New York.

Kharchenko, V.S. (1996). *Theory of defect-tolerant digital systems with version redundancy.* Military University, Kharkiv (In Russian).

Kharchenko, V.S. (1999). Methods of an Estimation of Multiversion Safety Systems. *Proceeding of the 17th International System Safety Conference*, Orlando, FL, Aug. 16-21, 1999.

Kharchenko, V.S., Sklyar, V.V. and Vilkomir, S.A. (2000). Models fitting of software reliability assessment for critical application systems. *Control systems and machines*, **3**, 59-69 (In Russian).

Laprie, J.-C. (1998). *Dependability Handbook.* Laboratory for Dependability Engineery, LAAS.

Lyu, M.R. (edit.) (1996). *Handbook of Software Reliability Engineering.* McGraw-Hill.

Schneidewind, N.F. (1997). Reliability Modeling for Safety-Critical Software. *IEEE Transactions on Reliability*, **46(1)**, 88-98.