HIERARCHICAL DECOMPOSITION FOR ESTIMATING RELIABILITY OF THE FAULT – TOLERANT SOFTWARE IN MISSION – CRITICAL SYSTEMS

KATERINA GOŠEVA – POPSTOJANOVA, AKSENTI GRNAROV
Department of Computer Science, Faculty of Electrical Engineering
PO Box 574, 91000 Skopje, Macedonia

ABSTRACT

This paper presents a hierarchical modeling approach aimed at reliability assessment over a mission period of the software fault tolerance technique based on N version programming. The model is constructed in three layers wherein submodels represent different parts of an object system and time scale distinctions. Our modeling approach is systematic as opposed to the ad hoc methods used in related works. Moreover, it permits modifications to be flexibly made at a specific level or in a specific submodel. Thus, the work presented here generalizes our previous work as it allows to consider general distributions of the versions time to failure and execution time at the first level. Also, at the third level, instead of the performability model in our previous work, we develop a new model aimed at reliability assessment over a mission period which supports the evaluation of a reliability over mission period in terms of the functional and timeliness requirements.

Keywords: Software fault tolerance, operational environment, failures of functionality, failures of performance, correlation, mission reliability.

1. INTRODUCTION

In this paper we analyze the software fault tolerance technique based on N version programming, first proposed in [1]. It relies on the application of design diversity: program versions are independently designed to meet the same system requirements. A consistent set of inputs is supplied to all versions and all N versions are executed in parallel. A decision mechanism must gather the available results from the N versions and determine the result to be delivered to the user.

A number of papers devoted to experimental and modeling based analysis of the software fault tolerance have appeared in literature. In the experimental studies diversity has been introduced in the form of different specifications [2], [3], [4], different programming languages [3], and for different distributions of test values over the input space [3]. All versions were developed independently by different teams, in some studies even by geographically distinct participants [3], [5], [6], [7]. Examining the results obtained by these experiments reveals several characteristics of NVP. First, the assumption of independence of failures between independently developed programs does not hold. Next, the coincident failures were observed in every experiment conducted thus far. At last, the failure behavior is very sensitive to the distribution of test values over the input space.

The papers devoted to the dependability modeling of software fault tolerance may be conveniently grouped into two classes. On one side, the major goal for the first class is the modeling of the dependability measures of the particular fault tolerant structure. Methods of specifying the system structure include combinational [8], [9], discrete time Markov chain [10], continuous time Markov process [11], [12], [13], fault trees and Markov reward models [14], extended stochastic Petri net [15], and generalized stochastic Petri nets [16] model types. On the other side, the major goal of models that belong to the second class is the precise meaning of the independence referred to the failure behavior of the diverse program versions [17], [18], [19].

The recently proposed performability models of the NVP in [20] and [21] place certain constraints on how properties affecting performance interact with those affecting dependability.

The fault tolerant software system investigated here is for the real – time mission – critical applications. We consider the systems for which it is not possible to perform a repair during their mission. These systems are characterized by high reliability requirements and stringent deadlines. The success of such a system depends not only on its logical correctness, but also on its timing correctness. In the mission – critical applications a software periodically gets the inputs from the environment, updates its internal states based on those inputs, and generates control output. It means that a mission is composed of a series of runs of fault tolerant software. The control stimulii are generated either by completing the execution or by timers when the run lasts beyond a pre – set maximum duration.

The model presented in this paper keeps the solution efficient by using hierarchical decomposition. It is constructed in three layers as shown in Fig. 1. The arrows indicate the interaction between submodels and the flow of information. Our modeling approach is systematic, more

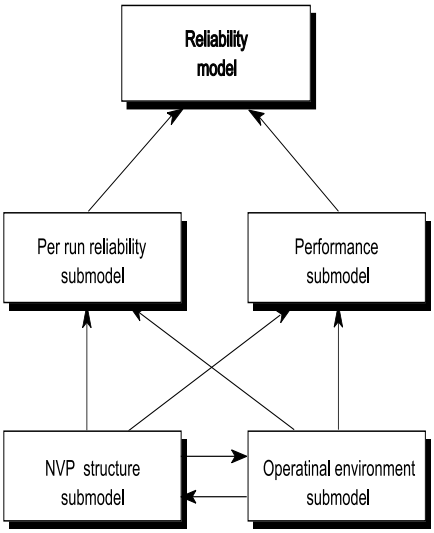

Figure 1: HIERARCHICAL MODEL

general and much more realistic then earlier models as it allows:

- to overcome the main modeling difficulty that arises from the correlation between version failures and execution times by distinguishing the NVP structure from its operational environment at the first layer
- to develop user – oriented model of the operational environment based on the concept of operational profile already used in software testing
- to integrate information on the software structure and on the system usage in its environment thus integrating the two disjoint modeling approaches from the previous models
- to derive reliability over a mission period in terms of the functional and timeliness requirements which is particularly relevant for real – time applications because failing to meet deadline have an adverse effect on system reliability.

2. NVP MODEL

We have developed the particular submodels that represent different parts of an object system and make time scale distinctions.

2.1. NVP STRUCTURE SUBMODEL

The NVP structure submodel considers NVP failure and execution behavior for a given input state. The time is treated locally measured from the beginning of a run. This NVP submodel is based on the following assumptions:

- The versions behave conditionally independent given a particular input state.
- Times to failure of program versions for given input state are identically distributed random variables with pdf $f_F(t;\upsilon)$ depending on d dimensional parameter vector $\upsilon = (\upsilon_1, \ldots, \upsilon_d)$.
- Execution times of program versions for given input state are identically distributed random variables with pdf $f_E(t;\psi)$ depending on b dimensional parameter vector $\psi = (\psi_1, \ldots, \psi_b)$.
- Due to the real – time constraint the system must make correct responses within a specified time interval $\tau > 0$.

The first assumption allows NVP failure and execution behavior for given input to be modeled from the single versions behavior. If the version produces output within specified time τ with its contents corrupted then *a functional failure* occurs. If the output (either correct or incorrect) is produced later than specified deadline τ then the version is said to have suffered *a performance (timing) failure.* In other words, a potential failure of functionality could be masked by a performance failure.

The distribution function $F_F(t;\upsilon)$ gives the probability that a single version will fail before t and it considers the failures of functionality. Since the duration of the execution time t is a random variable with density $f_E(t;\psi)$ we use the method described as randomized time [22] in order to derive distributions that take into account failures of performance as well. Thus, the probability that a single version produces functionally incorrect result before τ becomes

$$P_{fe}(\tau;\upsilon,\psi) = \int_0^\tau F_F(t;\upsilon) f_E(t;\psi) dt \qquad (1)$$

and the probability that a single version produces correct result before τ is given by

$$P_e(\tau;\upsilon,\psi) = \int_0^\tau [1 - F_F(t;\upsilon)] f_E(t;\psi) dt. \qquad (2)$$

The failure of performance (timing failure) occurs if a single version do not complete the execution until deadline τ

$$P_{ne}(\tau;\psi) = 1 - \int_0^\tau f_E(t;\psi) dt = 1 - F_E(\tau;\psi). \qquad (3)$$

Next, we consider the probabilities which characterize NVP failure and execution behavior for given input state. Thus, we define *timing failure* of N version system ($n = 2m-1$) for given input state to be the event that majority of versions do not produce output in time $\leq \tau$

$$P_{tf}(\tau;\psi) = \sum_{n_3=m}^{n} \binom{n}{n_3} P_{ne}^{n_3}(\tau;\psi) \left[1 - P_{ne}(\tau;\psi)\right]^{n-n_3}. \qquad (4)$$

If the majority of versions have completed the execution before τ it is possible that there is:
a majority of incorrect results (*functional failure*)

$$P_{ff}(\tau;\upsilon,\psi) = \sum_{n_2=m}^{n} \binom{n}{n_2} P_{fe}^{n_2}(\tau;\upsilon,\psi) \left[1 - P_{fe}(\tau;\upsilon,\psi)\right]^{n-n_2} \qquad (5)$$

a majority of correct results (*success*)

$$P_{ok}(\tau;\upsilon,\psi)=\sum_{n_1=m}^{n}\binom{n}{n_1}P_e^{n_1}(\tau;\upsilon,\psi)\left[1-P_e(\tau;\upsilon,\psi)\right]^{n-n_1} \quad (6)$$

or there is no majority of either correct or incorrect results

$$P_{nm}(\tau;\upsilon,\psi)=1-P_{tf}(\tau;\psi)-P_{ok}(\tau;\upsilon,\psi)-P_{ff}(\tau;\upsilon,\psi).$$

To the best of our knowledge our approach is the only one allowing analytical modeling of reliability that depends both on failures of functionality and on failures of performance. So far, this issue has been addressed only in the simulation based method presented in [15].

2.2. OPERATIONAL ENVIRONMENT SUBMODEL

The operational environment submodel considers the influence of the operational environment on versions failures and execution times. The concept of the operational environment, already used in software testing [23], is reviewed next. The operation of a software is broken down into series of *runs* and each run performs mapping between a set of input variables and a set of output variables and consumes a certain amount of execution time. Runs that are identical repetitions of each other are said to form *a run type*. Because the probabilities of occurrence of input states are the natural way of representing the program usage in its operational environment *the operational profile* is defined as a set of relative frequencies of occurrence of the run types.

We make the following additional assumptions:

- The environment is homogeneous or time invariant.
- The operational period is sufficiently long so the input state selection probabilities can be characterized by a steady state.
- The input states occur randomly and independently according to the operational profile.

Since the failure and execution behavior are quite sensitive to just how a system is utilized we need to take into consideration the change of the parameter vectors υ and ψ for different run types (input states). Therefore, the parameter vectors υ and ψ appear as random vectors Υ and Ψ respectively. The pair of random vectors $(\Upsilon\Psi) = (\Upsilon_1, \ldots, \Upsilon_d, \Psi_1, \ldots, \Psi_b)$ may be thought of as an event defined on a sample space $\mathcal{R}^{d+b}$ with $d+b$ dimensional distribution function

$$H_{\Upsilon\Psi}(\upsilon,\psi)=P\{\Upsilon_1\leq\upsilon_1,\ldots,\Upsilon_d\leq\upsilon_d,\Psi_1\leq\psi_1,\ldots,\Psi_b\leq\psi_b\}$$

while b variate distribution

$$H_{\Psi}(\psi) = P\{\Psi_1 \leq \psi_1,\ldots,\Psi_b \leq \psi_b\}$$

is marginal distribution of the distribution $H_{\Upsilon\Psi}(\upsilon,\psi)$.

In order to obtain numerical results it is possible to make assumptions and to use some theoretical distribution functions for $H_{\Upsilon\Psi}(\upsilon,\psi)$. Instead we choose to develop *the user – oriented model of operational environment*. Therefore, we partition the input space Ω by grouping run types that exhibit as nearly as possible homogeneous failure and execution behavior into *run categories*. Suppose that input space Ω is partitioned into run categories $A_{kj},\ 1 \leq k \leq r_1, 1 \leq j \leq r_2$ such that $\sum_k \sum_j A_{kj} = \Omega$. In this case the operational profile Q gives the probabilities $P\{\Upsilon = \upsilon^{\,k}, \Psi = \psi^j\} = p_{kj} = Q(A_{kj})$ that successive input states are chosen at random in a run category A_{kj}.

2.3. PER RUN RELIABILITY SUBMODEL

The per run reliability submodel treats the time locally measured from the beginning of a run. This submodel integrates the NVP structure submodel and operational environment submodel, thus accounting for the correlation between version due to the common input. We use the approach called stratification, which can also be described as randomization [22]. Therefore, we model parameter vectors υ and ψ as random vectors Υ and Ψ, so the probabilities $P_{ok}(\tau;\upsilon,\psi)$, $P_{ff}(\tau;\upsilon,\psi)$, and $P_{tf}(\tau;\psi)$ signify the conditional probabilities $P_{ok}(\tau|\Upsilon\Psi)$, $P_{ff}(\tau|\Upsilon\Psi)$, and $P_{tf}(\tau|\Psi)$. It follows that the unconditional per run probabilities for randomly chosen input state are given by:

$$P_{ok}(\tau) = \int_{\mathcal{R}^{d+b}} P_{ok}(\tau|\Upsilon = \upsilon, \Psi = \psi)\, dH_{\Upsilon\Psi}(\upsilon,\psi) \quad (7)$$

$$P_{tf}(\tau) = \int_{\mathcal{R}^{b}} P_{tf}(\tau|\Psi = \psi)\, dH_{\Psi}(\psi) \quad (8)$$

$$P_{ff}(\tau) = \int_{\mathcal{R}^{d+b}} P_{ff}(\tau|\Upsilon = \upsilon, \Psi = \psi)\, dH_{\Upsilon\Psi}(\upsilon,\psi) \quad (9)$$

where $P_{ok}(\tau|\Upsilon = \upsilon, \Psi = \psi)$, $P_{tf}(\tau|\Psi = \psi)$, $P_{ff}(\tau|\Upsilon = \upsilon, \Psi = \psi)$ are given in (6), (4), (5).

In the case of the user – oriented model of the operational environment Υ and Ψ take finite number of values, that is $H_{\Upsilon\Psi}(\upsilon,\psi)$ and $H_{\Psi}(\psi)$ are discrete distribution function and the relations (7), (8), and (9) signify the following

$$P_{ok}(\tau) = \sum_k \sum_j P_{ok}(\tau|\Upsilon = \upsilon^{\,k}, \Psi = \psi^j)\, p_{kj} \quad (10)$$

$$P_{tf}(\tau) = \sum_j P_{tf}(\tau|\Psi = \psi^j)\, p_j \quad (11)$$

$$P_{ff}(\tau) = \sum_k \sum_j P_{ff}(\tau|\Upsilon = \upsilon^{\,k}, \Psi = \psi^j)\, p_{kj} \quad (12)$$

where $p_j = P\{\Psi = \psi^j\} = \sum_k p_{kj}$ is a marginal distribution of a distribution $p_{kj} = P\{\Upsilon = \upsilon^{\,k}, \Psi = \psi^j\}$.

Distribution functions of form (10), (11), and (12) are called mixtures or compound distributions. Many methods that have been derived for estimating finite mixture parameters could be found in [24].

The formal analysis of the correlation between versions could be found in our earlier works [20], [25]. Here we emphasize that there is a correlation between versions behavior for a single input whenever Υ and Ψ vary for different run categories. Note that, our modeling approach is more general and much more realistic then previous ones since it considers the correlation between versions failure and execution behavior.

2.4. PERFORMANCE SUBMODEL

The performance submodel considers the execution behavior of the NVP and the events are distinguished only by their occurrences in time, independent of outcome result. This submodel treats time globally and represents the iterative nature of software's execution, that is the series of runs during the mission duration. At each run, the software accepts an input and produces an output that is a function only of the most recently accepted input.

For the performance submodel it suffices to consider a renewal process $\{N(t), t \geq 0\}$ where each run is represented by a renewal cycle. Let the time between successive renewals be such that T_i is elapsed time from $(i-1)$st run until the occurrence of ith run. We derive the distribution of T_i integrating the information contributed by the NVP structure submodel and operational environment submodel. First, we derive the conditional distribution for a given input state using the order statistics. The probability that at least m of the versions execution times for a particular input state lie in the interval $(0, t]$ is

$$F(t;\psi) = \sum_{l=m}^{n} \binom{n}{l} F_E^l(\tau;\psi)[1 - F_E(\tau;\psi)]^{n-l}. \quad (13)$$

Next, we obtain the unconditional probability distribution function of the time between successive renewals for random input state using the randomization procedure called stratification. As in the per run reliability submodel we treat the parameter vector ψ as random vector $\mathbf{\Psi}$, so (13) signifies the conditional probability distribution $F(t|\mathbf{\Psi})$. It follows that

$$F(t) = \int_{\mathcal{R}^b} F(t|\mathbf{\Psi} = \psi)\, dH_{\mathbf{\Psi}}(\psi). \quad (14)$$

For the user oriented model of the operational environment (14) becomes

$$F(t) = \sum_j F(t|\mathbf{\Psi} = \psi^j)\, p_j. \quad (15)$$

Finally, due to the real – time constraint, T_i is the time upon the completion of the NVP execution or upon reaching τ, whichever occurs first, that is

$$F_\tau(t) = \begin{cases} F(t), & \text{for } t < \tau \\ 1, & \text{for } t \geq \tau \end{cases} \quad (16)$$

with mean recurrence time $m_\tau = \int_0^\tau [1 - F(t)]dt.$ (17)

We emphasize that the versions execution times are correlated for a single input whenever the parameter vector $\mathbf{\Psi}$ is not identical for all run categories. In other words, we consider the correlation between execution times which is much more realistic than the assumption of independence made in [21] and [26].

2.5. RELIABILITY MODEL

The third layer supports the assessment of the NVP reliability over a mission. We derive the distribution of the time to failure as a function of global time using the decomposition of a renewal process. Therefore, we use a binary – valued random variable that distinguishes whether or not a specified service (in value and time) is performed properly for each run during a mission period. Consider a renewal process with distribution $F_\tau(t)$ defined by performance submodel (16), and suppose that each event is erased with probability $p = P_{ok}(\tau)$ computed by the per run reliability submodel (10). The resulting sequence of events constitutes a renewal process $\{\hat{N}(t), t \geq 0\}$ that registers only the successive occurrences of NVP failures. Its interval distribution is given by

$$\hat{F}(t) = \sum_{n=1}^{\infty} p^{n-1}\,(1-p)\, F_\tau^{n*}(t) \quad (18)$$

where F_τ^{n*} denotes the n-fold convolution of F_τ. If the Laplace transform of $F_\tau(t)$ is denoted by $\Phi(s)$ then the Laplace transform of the distribution of the time to failure $\hat{F}(t)$ becomes

$$\hat{\Phi}(s) = \sum_{n=1}^{\infty} p^{n-1}\,(1-p)\,\Phi^n(s) = \frac{(1-p)\,\Phi(s)}{1 - p\Phi(s)}. \quad (19)$$

The expression (19) in s – domain can be inverted numerically to obtain the solution in time domain, which leads to non trivial conservative estimates.

Another measure of interest is the mean time to failure (MTTF), that is the expected amount of time that software operates before failure. If only the value of MTTF is a desired solution, procedure is typically less complex due to the well known property of Laplace transform:

$$MTTF = E[\hat{F}(t)] = -\left.\frac{d\hat{\Phi}(s)}{ds}\right|_{s=0} = \frac{m_\tau}{1-p}. \quad (20)$$

3. NUMERICAL EXAMPLES

The objective of this section is to demonstrate the feasibility of the presented theoretical basis for the reliability analysis in the context of fault tolerant software in mission – critical systems. Due to the space limitations we restrict our attention only to the mean time to failure MTTF.

First, we use the exponential distributions for the time to failure and the execution time (Fig. 2) which are the usual assumptions in most software reliability models. In this case the parameter vectors consists of one component each $\Upsilon = (\lambda)$ and $\mathbf{\Psi} = (\mu)$. We assume the following values:

$\lambda_1 = 10^{-8}$ $\lambda_2 = 10^{-6}$ $\lambda_3 = 10^{-4}$ $\lambda_4 = 10^{-2}$
$\mu_1 = 0.5$ $\mu_2 = 0.2$ $\mu_3 = 0.125$

As a result, there are $r = 12$ different run categories. The operational profile is defined by two variate pmf $p_{kj} = P\{\Upsilon = \lambda_k, \mathbf{\Psi} = \mu_j\}$ that gives the probability that a successive input states are chosen at random in the run category defined by failure rate λ_k and execution rate μ_j. Different operational profiles (shown in Table 1) select different run categories A_{kj} with probabilities $p_{kj} = Q(A_{kj})$. The assumed timing constraint is $\tau = 30$ *msec*.

The operational profile $P1$ encounters inputs that re-

$P1$			$P2$			$P3$			$P4$			$P5$		
1	0	0	0.999	0	0	0.98	0	0	0.999	0	0	0.999	0	0
0	0	0	0	0.001	0	0	0.02	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0.001	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001

Table 1: DIFFERENT OPERATIONAL PROFILES

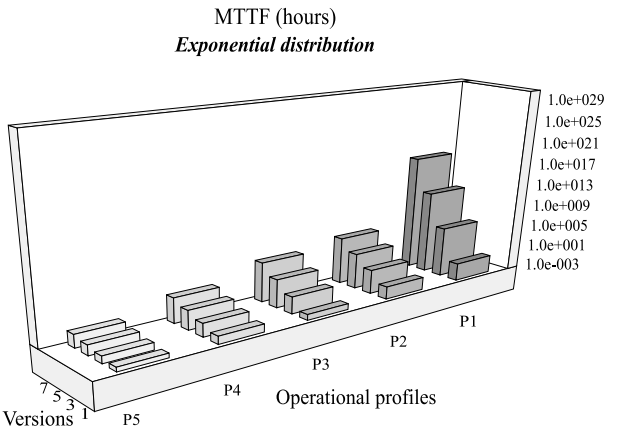

Figure 2: MTTF FOR EXPONENTIAL DISTRIBUTION

sult in the small versions execution period compared to the time to failure ($\mu \gg \lambda$). The versions failures and execution times are uncorrelated since there is no variation of the failure and execution rates over the input space. It is evident that in this case NVP substantially increases the MTTF. For example, three version system increases the MTTF by approximately six orders of magnitude relative to that of a single version. The operational profile $P2$ also encounters inputs that result in $\mu \gg \lambda$, but in this case the versions failures and execution times are correlated. It can be seen that seven versions would be required to achieve the same level of MTTF as in the case of three version system under the assumptions of independence. A slight modification of the operational profile for the same program characteristics ($P3$) leads to further reduction of the MTTF. The operational profile $P4$ assigns the same probability 0.001 as $P2$ to the worse program characteristics (higher failure rate and smaller execution rate). As a result, there is a further decrease in the estimated MTTF over that exhibited for operational profile $P2$. Although it is highly unlikely, we choose the operational profile $P5$ that selects inputs associated with even worse program characteristics ($\mu \sim \lambda$) which implies an exceedingly low MTTF.

The assumption that the distribution of the time to failure is exponential is far more realistic than the exponential distribution of the execution time. So, we choose to illustrate the influence of different distributions using the truncated normal distribution ($t \geq 0$) for versions execution time and the exponential distribution for the time to failure. In this case the parameter vectors are $\Psi = (\lambda)$ and $\Psi = (m, \sigma)$ with the following values:

$\lambda_1 = 10^{-8}$ $\lambda_2 = 10^{-6}$ $\lambda_3 = 10^{-4}$ $\lambda_4 = 10^{-2}$
$m_1 = 2$ $\sigma_1 = \sqrt{2}$
$m_2 = 5$ $\sigma_2 = \sqrt{5}$
$m_3 = 8$ $\sigma_3 = \sqrt{8}$

The operational profiles select with the same probability the inputs states associated with the same means λ and m as in the case of the previous example in order to make meaningful comparisons. It can be seen in Fig. 3 that the truncated normal distribution results in the higher MTTF compared to exponential distribution. As in the case of exponential distribution MTTF is improved with number of versions. In fact, increasing the number of versions offers higher improvement of the MTTF. Even the operational profile $P5$ results in a substantially higher MTTF over exponential distribution.

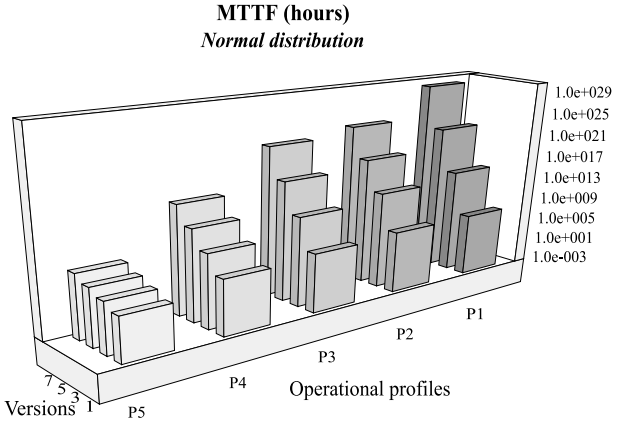

Figure 3: MTTF FOR TRUNCATED NORMAL DISTRIBUTION

There are some general remarks for all distribution functions. It is clear that redundancy alone does not guarantee fault tolerance. The degree of improvement depends both on the program characteristics (distribution functions and parameters values) and on the operational profile. The operational profiles that encounter inputs associated with comparatively high per run failure probabilities imply dramatic reduction of the MTTF which can not be significantly improved by any degree of fault tolerance. This is due to the fact that although for this choice of parameters there is no significant performance reduction, the increased failure probability results in greater chance to encounter unsuccessful run over a mission.

4. CONCLUSIONS

This paper presents the modeling based study of the software fault tolerance technique based on N version programming. The model addresses and resolves a number of basic issues and provides considerable insight into the NVP effectiveness and limitations. We keep the solution efficient by employing a hierarchical decomposition wherein submodels represent different parts of an object system and time scale distinctions. Our modeling approach is systematic as opposed to the ad hoc methods used in related works. Moreover, it is more general and much more realistic than earlier models since it

- considers general distributions of the time to failure and execution time
- integrates the NVP structure and system usage in its operational environment
- considers the correlation between versions failure and execution behavior for a single input
- results in the mission reliability that accounts for performance requirements.

Anticipating the future work, it would be useful to consider several possible extensions, such as the correlation of successive input states in order to model the phenomena of failure clustering.

References

[1] A.Avižienis, L.Chen, On the Implementation of N Version Programming for Software Fault Tolerance during Program Execution, *Proc. COMPSAC 77*, 1977, 149 – 155.

[2] A.Avižienis, J.Kelly, Fault Tolerance by Design Diversity: Concepts and Experiments, *IEEE Computers*, Aug 1984, 67 – 80.

[3] P.Bishop et al. Project on Diverse Software - An Experiment in Software Reliability, *Proc. 4th IFAC Workshop SAFECOMP*, 1985, 153 – 158.

[4] J. Kelly, T. McVittie, W. Yamamoto, Implementing Design Diversity to Achieve Fault Tolerance, *IEEE Software*, Jul 1991, 61 – 71.

[5] D.E.Eckhardt et al, An Experimental Evaluation of Software Redundancy as a Strategy For Improving Reliability, *IEEE Trans. on Software Engineering*, 17(7), 1991, 692 – 702.

[6] J.C.Knight, N.G.Leveson, An Experimental Evaluation of the Assumption of Independence in Multiversion Programming, *IEEE Trans. on Software Engineering*, SE-12(1), 1986, 96 – 109.

[7] M.R.Lyu, Yu – Tao He, Improving the N Version Programming Process through the Evaluation of a Design Paradigm, *IEEE Trans. on Reliability*, 42(2), 1993, 179 –189.

[8] A.Grnarov, J.Arlat, A.Avižienis, On the performance of Software Fault Tolerance Strategies, *Proc. 10th IEEE Int'l Symp. Fault Tolerant Computing*, 1980, 251 – 253.

[9] R.Scott, J.Gault, D.McAllister, Fault Tolerant Software Reliability Modeling, *IEEE Trans. on Software Engineering*, 13(5), 1987, 582 – 592.

[10] J.Arlat, K.Kanoun, J.Laprie, Dependability Modeling and Evaluation of Software Fault Tolerant Systems, *IEEE Trans. on Computers*, 39(4), 1990, 504 – 513.

[11] K.Goševa – Popstojanova, A.Grnarov, A New Markov Model of N Version Programming, *Proc. 2nd IEEE Int'l Symp. Software Reliability Engineering*, 1991, 210 – 215.

[12] K.Goševa – Popstojanova, A.Grnarov, N Version Programming with Majority Voting Decision: Dependability Modeling and Evaluation, *Proc. Euromicro 93*, 1993, 811 – 818.

[13] K.Goševa – Popstojanova, A.Grnarov, Dependability Modeling and Evaluation of Recovery Block Systems, *Proc. 4th IEEE Int'l Symp. Software Reliability Engineering*, 1993, 112 – 120.

[14] J.B.Dugan, M.R.Lyu, System Reliability Analysis of N Version Programming Application, *Proc. 4th IEEE Int'l Symp. Software Reliability Engineering*, 1993, 103 – 111.

[15] R.Geist, A.J.Offult, F.C. Harris Jr, Estimation and Enhancement of Real Time Software Reliability through Mutation Analysis, *IEEE Trans. on Computers*, 41(5), 1992, 550 – 558.

[16] K.Kanoun et al. Reliability Growth of Fault Tolerant Software *IEEE Trans. on Reliability*, 42(2), 1993, 205 –219.

[17] D.E.Eckhardt, L.D.Lee, A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors, *IEEE Trans. on Software Engineering*, SE-11(12), 1985, 1511 – 1517.

[18] V.F.Nicola, A.Goyal, Modeling of Correlated Failures and Community Error Recovery in Multiversion Software, *IEEE Trans. on Software Engineering*, 16(3), 1990, 350 – 359.

[19] L.A.Tomek, J.K.Muppala, K.S.Trivedi, Modeling Correlation in Software Recovery Blocks *IEEE Trans. on Software Engineering*, 19(11), 1993, 1071 – 1086.

[20] K.Goševa – Popstojanova, A.Grnarov, Performability Modeling of N Version Programming Technique, *Proc. 6th IEEE Int'l Symp. Software Reliability Engineering*, 1995, 209 – 218.

[21] A.T.Tai, J.F.Meyer, A.Avižienis, Performability Enhancement of Fault Tolerant Software, *IEEE Trans. on Reliability*, 42(2), 1993, 227 –237.

[22] W. Feller, *An Introduction to Probability and Its Applications* Volume II, John Wiley & Sons, 1971.

[23] J.D.Musa, Operational Profiles in Software Reliability Engineering, *IEEE Software*, Mar 1993, 14 – 32.

[24] B.S.Everitt, D.J.Hand, *Finite Mixture Distributions*, Chapman and Hall, 1981.

[25] K.Goševa – Popstojanova, A.Grnarov, N Version Programming: An Unified Modeling Approach, *Proc. EUROMICRO-22*, 1996, 363 – 370.

[26] M.Vouk, A.M.Paradkar, D.F.McAllister, Modeling Execution time of Multi – Stage N Version Fault Tolerant Software, *Proc. IEEE Computer Software and Applications Conference*, 1990, 505 – 511.