Effects of Failure Correlation on Software in Operation *

Katerina Goševa - Popstojanova †, Kishor Trivedi
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708-0294
E-mail: {katerina, kst}@ee.duke.edu.mk

Abstract

Since the early 1970's a number of models have been proposed for estimating software reliability. However, the realism of many of the underlying assumptions and the applicability of these models continue to be questioned. Our research work was motivated by the fact that although there are practical situations in which the assumption of independence among successive software failures could be easily violated, much of the published literature on software reliability modeling does not seriously address this issue. In this paper we present a modeling framework based on Markov renewal processes which naturally introduces dependence among successive software runs and enables the phenomena of failure correlation to be precisely characterized. Thus, incorporating failure correlation into dependability and performability predictions contributes toward more realistic modeling of software systems in operation.

1. Introduction

A number of analytical models have been proposed to address the problem of quantifying the software reliability. Most of them are focused on estimating software reliability based on its failure history, either during its debugging phase or during its validation phase [15], [5] [4], [12], [18]. These models treat the software as black box, assuming some parametric model of the time between failures or of the number of failures over a finite time interval. On the other hand, so called white box models that account for the software structure [9] are mostly restricted to the operational phase of the software life cycle.

One of the common assumptions made by most software reliability models is the assumption of independence among successive failures. In [13] it is emphasized that although there is an evidence that failures occur in groups, that is, there are some dependencies, the assumption of independence is made to cast the models into mathematically tractable form and simplify the estimation of model parameters. However, one of the reasons "why conventional reliability theory fails for software", listed in [10], is that the software runs are not always independent. The independence of successive software runs can be affected by operational conditions such as the independence of input sequence, the extent to which internal state of a software has been affected, and the nature of operations undertaken for execution resumption [11]. For instance, in many applications, such as real-time control systems, the sequence of input values to the software are correlated in time due to physical constraints. For these reasons, given a failure of a software for particular input, there is a greater likelihood of it failing for successive inputs. In applications that operate on demand, similar types of demands made on software tend to occur close to each other which can result in a succession of failures.

To summarize, although there may be dependencies among successive software runs which can result in failure clustering, only a few published papers consider this issue. These include the empirically developed Fourier series model [2] and the Compound-Poisson software reliability model [16]. Most of the existing models of fault-tolerant software also assume independence among successive runs. The only exceptions are the models presented in [3] and [17].

We have recently proposed a software reliability modeling framework that is capable of incorporating the possible dependence among successive software runs, that is, the effect of failure clustering. In [7] we have demonstrated its use for testing phase and validation phase. The goal of this paper is to extend our previous work [8] by appling the model to the operational phase when the software's ability to deliver a proper service is preserved and the reliability is stable. The use of Markov renewal processes leads to both conceptual simplification in building the model and compu-

*Supported in part by NSF under the grant EEC-9714965, Department of Defense, Telcordia, and Alcatel as a core project in the Center for Advanced Computing and Communication.

†On leave of absence from the Dept. of Computer Science, Faculty of Electrical Eng., University "Sv. Kiril i Metodij", Skopje, Macedonia.

tational simplification in deriving a number of measures that quantify the software behavior, such as reliability, availability and performability. In addition, the proposed approach allows a great flexibility in generalizing the model; it can easily be extended to cover several key areas, such as real-time software and fault-tolerant software.

The paper is organized as follows. The conceptual model is given in Section 2. Section 3 presents our modeling framework based on Markov renewal processes. In Section 4 we study software reliability. Next we derive two different performability measures in Section 5 and the expression for availability in Section 6. The possible extensions and generalizations of the presented model are discussed in Section 7. Finally, the concluding remarks are given in Section 8.

2. Conceptual model

First consider in some detail the concept of software runs. The operation of software can be broken down into a series of runs [12]. Each run performs mapping from a set of input variables to a set of output variables and consumes a certain amount of execution time. Even for the software that operates continuously it is still possible and more convenient to divide the operation into runs by subdivision of time associated with some user-oriented tasks [12]. The information associated with software runs can generally be grouped into two categories:

- *Input and Outcome.* The descriptive information about each specific run generally specifies the input, the operational environment, and the outcome that has been obtained (success or failure).
- *Timing.* This includes specific time associated with each run, such as start time, normal termination time for successful runs, or failure time for failed runs.

Since each software run has two possible outcomes, success and failure, the standard way of looking at the sequence of software runs is to consider it as a sequence of independent Bernoulli trials, where each trial has probability of success p and probability of failure $q = 1 - p$. The two distribution functions, binomial and geometric, are connected with the independent Bernoulli trials:

- number of runs that have failed among n successive software runs has the binomial *pmf*
- number of software runs between two failures has the geometric *pmf*.

Since the number of software runs n is large and the failure probability $q = 1 - p$ is small, the well known limiting results for the binomial and geometric *pmf* are usually used as basic assumptions in software reliability models:

- number of failures in the limit has the Poisson distribution
- time between failures in the limit is exponentially distributed.

The work presented in this paper attempts to extend the classical software reliability theory in order to consider a sequence of possibly dependent software runs. The sequence of successive software runs (successful or failed) can be viewed as a realization of point events in time, that is, as a point process. Poisson process is the simplest point process. It considers only the failure points, that is, disregards the successful runs between two failures and ignores the information conveyed by them. In this paper we view the sequence of dependent software runs, when the outcome of each run depends on the outcome of the previous run, as a sequence of dependent Bernoulli trials. Therefore, we need to consider both failed and successful runs. A convenient way of specifying a sequence of more than one class of points in continuous time is the Markov renewal process.

3. Modeling framework based on Markov renewal processes

Consider a process constructed as follows. First take a k-state homogeneous discrete time Markov chain (DTMC) with transition probability matrix $P = [p_{ij}]$. Next construct a process in continuous time by making the time spent in a transition from state i to state j have distribution function $F_{ij}(t)$, such that times are mutually independent. Such a process is a semi Markov process (SMP). The family of stochastic processes used in this paper, called Markov renewal process (MRP), may be shown to be equivalent to the family of SMP [1]. Thus, the SMP records the state of the process at each time point t, while the MRP is a point (counting) process which records the number of times each of the possible states has been visited up to time t. If one regards the MRP as consisting of k dependent renewal processes $N_1(t), \ldots, N_k(t)$, where $N_i(t)$ refers to the points of class i, the observed process of points is the superposition $N(t) = N_1(t) + \ldots + N_k(t)$.

The Markov renewal model formulation allows great flexibility in both building and solving the model. Thus, we build the model in two stages, describing intuitively and separately the two elements of randomness in software operation: the uncertainty about the outcome of the software run and the uncertainty about the time that takes the run to be completed. First we define a DTMC which considers the outcomes from the sequence of possibly dependent software runs in discrete time. Next, we construct the process in continuos time by attaching the distributions of the run's execution time to the transitions of the DTMC turning it into an MRP.

3.1. Model in discrete time

We view the sequence of software runs in discrete time as a sequence of dependent Bernoulli trials in which the probability of success or failure at each trial depends on the outcome of the previous trial. Let us associate with the i-th software run a binary valued random variable

$$Z_i = \begin{cases} 0 & \text{denotes a success on the } i\text{-th run} \\ 1 & \text{denotes a failure on the } i\text{-th run.} \end{cases}$$

Suppose that if i-th run results in failure the probability of failure and the probability of success at the $(i+1)$-st run are $P\{Z_{i+1} = 1|Z_i = 1\} = q$ and $P\{Z_{i+1} = 0|Z_i = 1\} = 1 - q$ respectively. Similarly, if i-th run results in success then there are probabilities $P\{Z_{i+1} = 0|Z_i = 0\} = p$ and $P\{Z_{i+1} = 1|Z_i = 0\} = 1-p$ of success and failure respectively at the $(i+1)$-st run. A very plausible assumption for operational software is that the probabilities of failure $1-p$ and q are much smaller than the probabilities of success p and $1-q$. A program for which this assumption could not be made would be enormously unreliable, and it is unlikely that its use would be ever attempted.

The sequence of dependent Bernoulli trials $\{Z_i, i \geq 1\}$ defines a discrete time Markov chain with two states shown in Figure 1. One of the states, denoted by 0, is regarded as success and the other, denoted by 1, as failure. Accordingly, transition probability matrix is given by

$$P = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix}. \tag{1}$$

Since p and q are probabilities, it follows that $0 \leq p, q \leq 1$ and $|p + q - 1| \leq 1$. However, the boundary cases, when the equalities hold, are excluded from further analysis since they are somewhat trivial with no practical interest. Thus, we impose the condition $0 < p, q < 1$ on transition probabilities, which implies that $|p + q - 1| < 1$. In other words, the Markov chain is irreducible and aperiodic, with all states recurrent nonnull.

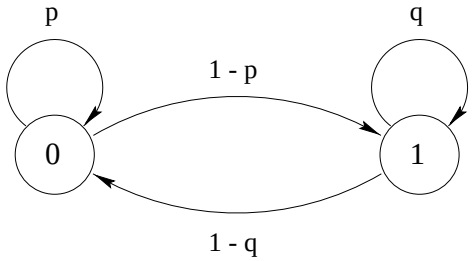

Figure 1. Markov interpretation of dependent Bernoulli trials

First consider in some detail the properties of the Markov chain. Since we consider the operational usage of software when the reliability is stable, it is assumed that the DTMC is stationary in the wide sense [1]. It means that the mean $E[Z_i]$ is constant

$$E[Z_i] = P\{Z_i = 1\} = \theta \tag{2}$$

and autocovariance is a function of k only

$$Cov[Z_i, Z_{i+k}] = E[Z_i Z_{i+k}] - (E[Z_i])^2. \tag{3}$$

It is obvious that θ is the unconditional per run failure probability. Thus, we have

$$\begin{aligned} P\{Z_{i+1} = 1\} =& P\{Z_{i+1} = 1|Z_i = 0\}\, P\{Z_i = 0\} \\ &+ P\{Z_{i+1} = 1|Z_i = 1\}\, P\{Z_i = 1\}. \end{aligned}$$

The stationarity of the chain (2) imposes that $\theta = (1-p)(1-\theta) + q\theta$, that is,

$$\theta = \frac{1-p}{2-p-q}, \qquad 0 < \theta < 1. \tag{4}$$

The correlation coefficient between outcomes of the software k runs apart is specified by the autocorrelation function $\beta_k = Cov[Z_i, Z_{i+k}]/Var[Z_i]$. It can be shown that the correlation coefficient between outcomes of two successive runs is given by

$$\beta_1 = \frac{q-\theta}{1-\theta}. \tag{5}$$

Using (4) it follows that the autocorrelation function of lag 1 given by (5) can be expressed as

$$\beta_1 = p + q - 1, \qquad -1 < \beta_1 < 1. \tag{6}$$

Deriving one step probabilities p and q as functions of θ and β_1 (given by (4) and (6) respectively), transforms the DTMC transition probability matrix (1) into

$$P = \begin{bmatrix} (1-\theta) + \theta\,\beta_1 & \theta - \theta\,\beta_1 \\ (1-\theta) - (1-\theta)\,\beta_1 & \theta + (1-\theta)\,\beta_1 \end{bmatrix}. \tag{7}$$

It is obvious from equation (6) that $\beta_1 = 0$ if $p + q = 1$, that is, each run can fail with probability $q = 1 - p$ independently of the outcome of previous run. In this case the Markov chain describes a sequence of independent Bernoulli trials. If $\beta_1 \neq 0$ then the DTMC describes the sequence of dependent Bernoulli trials and enables us to accommodate possible dependence among successive runs. Depending on the relation between the conditional probabilities of failure $1-p$ and q we can describe either presence or lack of failure clustering. Therefore, we keep $1-p$ fixed and let either $q > 1-p$ or $q < 1-p$. It follows that the model admits as special cases the following:

1. **Failures are independent** $(q = 1 - p)$**.** Each run has the same probability of failure, independently of the outcome of the previous run.

2. **A lack of clustering** $(q < 1-p)$**.** Successive software runs are negatively correlated $-1 < \beta_1 < 0$, that is, software failure is more probable after a success than after a failure.

3. **Failures occur in clusters** $(q > 1-p)$**.** Successive runs are positively correlated $0 < \beta_1 < 1$, that is, software failure is more probable after a failure than after a success.

3.2. Model in continuous time

The next step in the model construction is to obtain a process in continuous time by considering the time spent in a transition from state i to state j with a distribution function $F_{ij}(t)$. In our case $F_{ij}(t)$ are the distributions of the duration of software runs, that is, the time that takes software runs to be executed. It is assumed that software execution time T_{ex} has the same distribution $F_{ex}(t) = P\{T_{ex} \leq t\}$ (with finite mean $E[T_{ex}]$) regardless of the outcome, that is, $F_{ij}(t) = F_{ex}(t)$ for $0 \leq i,j \leq 1$. Considering the situation when software execution times are not identically distributed for successful and failed runs is straightforward and will be discussed in Section 7.

With the addition of $F_{ex}(t)$ to the DTMC, we obtain the model in continuous time, that is, an MRP. Thus, the sequence of software runs in continuous time constitutes a point process with two classes of points: success and failure. The total number of software runs $\{N(t), t \geq 0\}$ is a superposition of two dependent renewal processes $N_S(t)$ and $N_F(t)$ which refer to the number of times states 0 (success) and 1 (failure) of the DTMC have been visited in the interval $(0,t]$.

In what follows we derive a number of reliability, performability and availability measures and reveal how failure correlation influences each of the measures.

4. Software reliability modeling

In software reliability modeling only the points of particular type, i.e., failures are of interest. Therefore, it is necessary to consider only the distribution of an interval between successive failures $F_F(t)$ which is the interval distribution of the point process $N_F(t)$ that records the number of failures in the interval $(0,t]$. If we assume that the initial state is 0 with probability 1 then the point process $N_F(t)$ is a delayed renewal process with a time to first failure having different *CDF* from the times between successive failures.

We derive the distribution of the time between failures (inter-failure time) $F_F(t)$ in two steps. First consider the discrete random variable X defined as a number of runs between two successive visits to the failure state of the DTMC. Clearly, the random variable X has the *pmf*

$$P\{X=k\} = \begin{cases} q & \text{if } k=1 \\ (1-q)\,p^{k-2}\,(1-p) & \text{if } k \geq 2. \end{cases} \quad (8)$$

Next, consider the model in continuous time. It follows that the *CDF* of the inter-failure time $F_F(t) = P\{T_F \leq t\}$ is given by

$$F_F(t) = q\,F_{ex}(t) + \sum_{k=2}^{\infty} (1-q)\,p^{k-2}\,(1-p)\,F_{ex}^{k*}(t) \quad (9)$$

where F_{ex}^{k*} denotes the k-fold convolution of F_{ex}. The Laplace - Stieltjes transform (LST) of $F_F(t)$ is given by

$$\tilde{F}_F(s) = \frac{q\,\tilde{F}_{ex}(s) + (1-p-q)\,\tilde{F}_{ex}^2(s)}{1-p\,\tilde{F}_{ex}(s)} \quad (10)$$

where $\tilde{F}_{ex}(s)$ is the LST of $F_{ex}(t)$.

In a similar manner it can be shown that the distribution of the time to first failure has the LST given by

$$\tilde{F}_1(s) = \frac{(1-p)\,\tilde{F}_{ex}(s)}{1-p\,\tilde{F}_{ex}(s)}. \quad (11)$$

Depending on the particular distribution of the execution time $F_{ex}(t)$, the expression for $F_F(s)$ can be inverted either symbolically or numerically to obtain the solution for $F_F(t)$ in time domain. Reasonably simple closed-form results can be obtained when the distribution function of the run's execution time $F_{ex}(t)$ has rational LST, such as stage-type distributions discussed in [1]. In that case $\tilde{F}_F(s)$ is also a rational function and the inversion of (10) is in principle straightforward.

We now develop some general properties of the distribution of the time between failures. Due to the well known property of the LST, the moments can be derived easily by a simple differentiation of (10). It follows that the mean time to failure (*MTTF*) is given by

$$E[T_F] = -\left.\frac{d\tilde{F}_F(s)}{ds}\right|_{s=0} = \frac{2-p-q}{1-p}\;E[T_{ex}] \quad (12)$$

where $E[T_{ex}]$ is the run's mean execution time. Then by equation (4) we have

$$E[T_F] = \frac{1}{\theta}\;E[T_{ex}]. \quad (13)$$

This result has a simple physical interpretation when one accounts for the fact that $1/\theta$ is the mean number of runs between two failures $E[X]$. Thus, the *MTTF* is a product of the mean number of runs between two successive failures and the run's mean execution time[1].

[1]Note that $F_F(t)$ is a proper distribution function with finite mean $E[T_F]$ as a consequence of the assumptions that we made ($0 < p,q < 1$ and $0 < E[T_{ex}] < \infty$). Although it is trivial to consider the possible defective distributions of the times between failures (viz, the non-existence of MTTF), the discussion is omitted due to the space limitations.

If successive runs are independent ($q = 1 - p$), *MTTF* becomes $E[T_F^{in}] = E[T_{ex}]/(1-p)$ which means that we can rewrite (12) as

$$E[T_F] = (2 - p - q)\, E[T_F^{in}] = (1 - \beta_1)\, E[T_F^{in}]. \quad (14)$$

Also, we derive an expression for the variance

$$Var[T_F] = \frac{Var[T_{ex}]}{\theta} + \frac{(1-\theta)(1+\beta_1)}{\theta^2(1-\beta_1)}(E[T_{ex}])^2.$$

It follows that the squared coefficient of variation is

$$C^2_{T_F} = \frac{Var[T_F]}{(E[T_F])^2} = \frac{1}{1-\beta_1} C^2_{T_F^{in}} + \frac{\beta_1}{1-\beta_1}\,(1 - 2\theta) \quad (15)$$

where $C_{T_F^{in}}$ is the coefficient of variation for the case of independent successive runs. Since $\theta < 1/2$ will be satisfied for any software in operational phase, it follows that the sign of the second term of equation (15) will depend only on $\beta_1 = p + q - 1$.

The derived equations for the *MTTF* (14) and the coefficient of variation (15) enable us to study the effects of failure correlation on software reliability

- **A lack of clustering**

  When $\beta_1 < 0$ from (14) we get $E[T_F] > E[T_F^{in}]$.

  In this case the second term in (15) is negative and hence $C^2_{T_F} < C^2_{T_F^{in}}/(1-\beta_1) < C^2_{T_F^{in}}$.

  It follows that assuming a lack of clustering leads to greater mean and smaller variability of the inter-failure times compared to the independent case.

- **Failures occur in clusters**

  If, on the other hand, $\beta_1 > 0$ then $E[T_F] < E[T_F^{in}]$.

  Since the second term in (15) is positive it follows that $C^2_{T_F} > C^2_{T_F^{in}}/(1-\beta_1) > C^2_{T_F^{in}}$.

  These results imply that in the presence of failure clustering the inter-failure time has smaller mean and greater variability compared to the independent case. In other words, when failures are indeed clustered the independence assumption results in optimistic estimations.

We next focus on the special case when the distribution of the run's execution time is exponential so that $f_{ex}(t) = \mu e^{-\mu\, t}$, since it relates the MRP approach to the existing software reliability models. Inverting (10) leads to the *pdf* of the time between failures given by

$$f_F(t) = \frac{(1-q)}{p}\,(1-p)\mu e^{-(1-p)\mu\, t} + \frac{(p+q-1)}{p}\,\mu e^{-\mu\, t}. \quad (16)$$

Assuming independence among successive software runs ($q = 1 - p$) reduces (16) to

$$f_F^{in}(t) = (1-p)\,\mu\, e^{-(1-p)\mu\, t}. \quad (17)$$

It follows that the time between failures is exponentially distributed with rate $(1-p)\,\mu$ when the successive software runs are independent with exponentially distributed execution times. In this case the Markov renewal process reduces to the homogeneous Poisson process. Thus, the standard way of describing the failure behavior of a software system in operation is obtained as the simplest special case of the presented model.

5. Software performability modeling

Performability is a unified measure that combines the quantifications of reliability and performance, and reveals their effect on the ability of a system to complete a certain amount of useful work. We propose two different performability measures. In both cases the performability is defined with respect to the number of software runs that benefit the user during a bounded time period of duration t. In the case of the first measure if software fails at any run it is considered failed for the whole mission, that is, no run either prior or subsequent to such failure is beneficial. Thus, this measure is appropriate for mission-critical systems. The second performability measure is more suitable for high availability systems since the failures are counted only at the run in which they manifested themselves, which means that all successful runs during the given period are beneficial.

For the first performability measure $U(t)$ if a software fails at any run $N_F(t) > 0$ no run either prior or subsequent to such a failure is considered beneficial and hence $U(t) = 0$. It follows that $U(t)$ can be formulated as

$$U(t) = \begin{cases} N(t) & \text{if } N_F(t) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The probability that no run fails in n runs is derived using the Markov property

$$\begin{aligned} &P\{Z_n = Z_{n-1} = Z_{n-2} = \ldots = Z_2 = Z_1 = 0\} \\ &\quad = P\{Z_n = 0 | Z_{n-1} = 0\} \cdot \ldots \cdot P\{Z_2 = 0 | Z_1 = 0\} \\ &\quad \times P\{Z_1 = 0\} = p^{n-1}\,(1-\theta). \end{aligned}$$

It follows that the moment generating function of $U(t)$ is

$$E[e^{sU(t)}] = E[1 - (1-\theta)\, p^{N(t)-1} + e^{sN(t)}(1-\theta)\, p^{N(t)-1}]$$

and its expectation can be expressed as

$$\begin{aligned} E[U(t)] &= \left.\frac{dE[e^{sU(t)}]}{ds}\right|_{s=0} = E[N(t)\,(1-\theta)p^{N(t)-1}] \\ &= \frac{1-\theta}{p}\;\; E[U^{in}(t)] = \frac{1-\theta}{(1-\theta)+\theta\beta_1}\;\; E[U^{in}(t)] \end{aligned} \quad (18)$$

where $E[U^{in}(t)] = E[N(t)\, p^{N(t)}]$ is the benefit that can be expected if the successive software runs were independent.

It is clear that if $\beta_1 > 0$ then $(1-\theta)/[(1-\theta)+\theta\beta_1] < 1$, that is, if failures do occur in clusters then the performability $E[U(t)]$ is reduced by factor $(1-\theta)/[(1-\theta)+\theta\beta_1]$ compared to the independent case. On the other hand, a lack of clustering has just the opposite effect.

For the second performability measure failures are counted only at the run in which they manifested themselves, that is, all successful runs up to time t are beneficial. It follows that it can be defined as the expected number of successful runs in the time interval $(0,t]$

$$M(t) = E[N_S(t)] = \sum_{k=1}^{\infty} F_S^{k*}(t) \tag{19}$$

where F_S^{k*} denotes the k-fold convolution of F_S. The LST of (19) is given by

$$\tilde{M}(s) = \sum_{k=1}^{\infty} \tilde{F}_S^k(s) = \frac{\tilde{F}_S(s)}{1-\tilde{F}_S(s)}. \tag{20}$$

The distribution of the time between successful runs $F_S(t) = P\{T_S \le t\}$ which is the interval distribution of the renewal process $\{N_S(t), t \ge 0\}$ that registers only the successive visits to state 0 is derived in a similar manner as $F_F(t)$. Its LST is given by

$$\tilde{F}_S(s) = \frac{p\,\tilde{F}_{ex}(s) + (1-p-q)\,\tilde{F}_{ex}^2(s)}{1-q\,\tilde{F}_{ex}(s)}. \tag{21}$$

Substituting (21) into (20) leads to

$$\tilde{M}(s) = \frac{p\tilde{F}_{ex}(s) + (1-p-q)\tilde{F}_{ex}^2(s)}{1-q\tilde{F}_{ex}(s) - p\tilde{F}_{ex}(s) - (1-p-q)\tilde{F}_{ex}^2(s)}. \tag{22}$$

The expression (22) can be inverted either symbolically or numerically to obtain the solution for $M(t)$ in time domain. However, since the time interval $(0,t]$ is much greater than T_S, that is, many successful runs are expected to take place up to time t we can use the asymptotic form of the renewal function for large t ($t \to \infty$) [1]

$$M(t) \sim \frac{t}{E[T_S]}. \tag{23}$$

By simple differentiation of (21) we get $E[T_S] = E[T_{ex}]/(1-\theta)$. Hence

$$M(t) \sim (1-\theta) \cdot \frac{t}{E[T_{ex}]}. \tag{24}$$

The above result can be given a simple intuitive meaning; the second term in equation (24) is the expected number of software runs for large t. Multipling it by the per run probability of success $1-\theta$ gives us the expected number of successful runs.

Having in mind that the expected number of successful runs for the independent case is

$$M^{in}(t) \sim p\ \frac{t}{E[T_{ex}]}$$

it is clear that (24) can be rewritten as

$$M(t) \sim \frac{1-\theta}{p}\ M^{in}(t) = \frac{1-\theta}{(1-\theta)+\theta\beta_1}\ M^{in}(t). \tag{25}$$

It follows from (25) that the failure clustering ($\beta_1 > 0$) has the same effect on $M(t)$ as on the first performability measure: the expected number of successful run in $(0,t]$ is reduced by factor $(1-\theta)/[(1-\theta)+\theta\beta_1]$ compared to the independent case. Of course, (25) is a limiting result only, whereas (18) is exact.

6. Software availability modeling

An instantaneous availability is simply the probability that the software is operational at time t. Assuming that the initial state is 0, the availability is the probability that the semi Markov process is in state 0 at time t, which we denote by $A(t) = P_{00}(t)$. The semi Markov process will be in state 0 at time t, given it was in state 0 at $t = 0$, if one of the following mutually exclusive events occurs:

- no transition occurs in $(0,t]$ with probability $1-H_0(t)$, where $H_0(t) = pF_{ex}(t) + (1-p)F_{ex}(t) = F_{ex}(t)$ denotes the distribution of the amount of time until the next transition occurs given that the process has just entered 0
- the process returns to state 0 before t according to $F_S(t)$ and then, in the remaining time, ending up in state 0 according to $P_{00}(t)$.

Thus, $P_{00}(t) = 1-H_0(t)+\int_0^t P_{00}(t-x)dF_S(x)$ and hence the LST of the availability becomes

$$\tilde{A}(s) = \frac{1-\tilde{F}_{ex}(s)}{1-\tilde{F}_S(s)} = \frac{1-q\tilde{F}_{ex}(s)}{1+(1-p-q)\tilde{F}_{ex}(s)}. \tag{26}$$

If the distribution of the run's execution time $F_{ex}(t)$ is given, the above equation enables us to compute the instantaneous availability $A(t)$ as a function of time. Often we are interested in a steady-state availability A defined as the limiting value of $A(t)$ as t approaches infinity. To examine limiting form of $A(t)$ as $t \to \infty$ requires an analysis of $\tilde{A}(s)$ as $s \to 0$, that is,

$$A = \lim_{s\to 0} \tilde{A}(s) = \frac{1-q}{2-p-q} = 1-\theta. \tag{27}$$

Note that the steady-state availability would also depend on the run's mean execution times if they were different for successful and failed runs.

Since the availability for the independent case is $A^{in} = p$ it follows form (27) that when failures occur in clusters ($\beta_1 > 0$) the software availability is reduced by the same factor as performability $(1-\theta)/p = (1-\theta)/[(1-\theta)+\theta\beta_1]$.

7. Extensions and generalizations

The presented model can be generalized in many ways. We now discuss some of these generalizations, together with brief comments on the nature of new ideas involved.

i) Markov renewal process with more than two states

The two state DTMC describes a sequence of dependent Bernoulli trials and naturally generalizes the traditional way of looking at the sequence of software runs. Nevertheless, by adding suitable states to the DTMC we can account for failures with different severity, software maintenance or periods of time when the software is idle.

ii) Higher order of dependance

For the presented model we have assumed that the outcome of the next run to be executed Z_{i+1} will depend probabilistically on the present run only Z_i and is independent of the past history, that is, we assume that the embedded Markov chain is a first order chain. However, the hypothesis that the DTMC is of a given order needs to be tested. Thus, if the dependence goes back more than one run the system can be described by a model of higher order. In such cases the higher order Markov chain can be represented as a first order chain by redefining the state space appropriately [1]. This representation is useful because the results for the first order Markov chain can be carried over. However, the size of the state space grows fast with the order of the chain.

iii) Real-time software

In real-time applications each run is under a real-time constraint which takes the form of an upper bound on the time to complete a software execution. It means that T_{ex} will be the time upon the end of execution or upon reaching a deadline τ, whichever occurs first. It follows that run's execution time *CDF* will coincide with $F_{ex}(t)$ for $0 \leq t < \tau$, otherwise it will be equal to 1.

iv) Fault-tolerant software

The proposed modeling approach can be used for modeling fault-tolerant software (FTS) systems. Per run failure probability θ and run's execution time distribution $F_{ex}(t)$ for a particular FTS structure can be derived using a variety of existing FTS models (see [6] and references therein). Thus, in addition to the inter-version failure correlation on a single run considered in related works, our approach enables us to account for the correlation among successive failures.

v) Different CDF for the execution times of the successful and failed runs

If we assume that software execution times are not identically distributed for successful and failed runs then the distribution functions $F_{ij}(t)$ will depend of the type of point at the end of the interval. Thus, $F_{00}(t) = F_{10}(t) = F_{ex_S}(t)$ and $F_{01}(t) = F_{11}(t) = F_{ex_F}(t)$, where $F_{ex_S}(t)$ and $F_{ex_F}(t)$ are the distribution functions of the execution times of the successful T_{ex_S} and failed runs T_{ex_F}, respectively. By making appropriate changes in (9) we get the interval distribution of the failure process $N_F(t)$

$$F_F(t) = Pr\{T_F \leq t\} = qF_{ex_F}(t) + \sum_{k=2}^{\infty} (1-q)\, p^{k-2}\, F_{ex_S}^{(k-1)*}(t)\, (1-p)\, F_{ex_F}(t). \quad (28)$$

MTTF and variance of the inter-failure time can be derived in the same manner as in Section 4.

Now consider the time between events $\{T_1, T_2, \dots\}$ of the point process $N(t)$, where T_i indicates the execution time of i-th run. There are two types of intervals T_i with *CDF* $F_{ex_S}(t)$ and $F_{ex_F}(t)$ that occur in accordance with a DTMC transition probability matrix P given by (1). The interval distribution of the point process $N(t)$ is $F_T(t) = (1-\theta)F_{ex_S}(t) + \theta F_{ex_F}(t)$ where θ is given by (4).

The correlational properties of the sequence $\{T_i\}$ are described by the lag 1 autocorrelation function

$$\rho = \frac{Cov[T_i, T_{i+1}]}{Var[T]} \quad (29)$$

which can be shown to be equal to equation (30). The derivation is not given here due to the space limitations.

It follows from (30) that the execution times of successive software runs T_i are correlated if the outcome of each software run is dependent on the outcome of the previous run $\beta_1 \neq 0$, and the execution times of the successful and failed runs have different means $E[T_{ex_S}] \neq E[T_{ex_F}]$. In other words $\beta_1 \neq 0$ implies that the classes of successive events are dependent, but in order the times between events to be dependent the additional condition $E[T_{ex_S}] \neq E[T_{ex_F}]$ is necessary.

vi) Matrix form solution of the model

The model formulation and the derivation of the measures of interest presented in the paper are an intuitive and natural way to view the problem. However, this approach becomes too cumbersome if we consider the model which consists of DTMC with more states and different distribution functions $F_{ij}(t)$ attached to the transitions from state i to j. In that case it is reasonable to use the solution in matrix form. For detailed and extensive treatment the reader is referred to [14].

$$\rho = \frac{(E[T_{ex_S}] - E[T_{ex_F}])^2 \, (1-\theta)\, \theta}{(1-\theta)\, Var[T_{ex_S}] + \theta\, Var[T_{ex_F}] + (1-\theta)\, \theta\, (E[T_{ex_S}] - E[T_{ex_F}])^2} \cdot \beta_1 \tag{30}$$

8. Conclusion

The research work presented in this paper is devoted to the development of the modeling framework that considers a sequence of possibly dependent software runs and enables us to study the effects of failure correlation on reliability, performability and availability of the software system in operational phase. The proposed Markov renewal approach allows a great flexibility in building, solving and generalizing the model. Thus, we describe intuitively and separately the two elements of randomness in software operation: the uncertainty about the outcomes of the software runs and the uncertainty about the time that takes the runs to be completed. Using the theory of Markov renewal processes we have derived a number of dependability and performability measures that quantify software behavior. We have also presented a brief discussion of the possible extensions and generalizations of the presented model.

The aim of this paper is to extend the classical software reliability theory by incorporating failure correlation into the predictions which we believe has important practical implications. The obtained results clearly demonstrate that assuming independence among successive software runs will result in optimistic estimations when failures do occur in clusters. It follows that the effect of correlation among successive software failures has to be taken into account, especially in applications where the key issue is not to overestimate the dependability or performability. The subject of our future research is the applicability of the model to various examples of software systems in operation.

References

[1] D.R.Cox, H.D.Miller, *The Theory of Stochastic Processes*, Chapman and Hall, 1990.

[2] L.H.Crow, N.D.Singpurwalla, "An Empirically Developed Fourier Series Model for Describing Software Failures" *IEEE Trans. on Reliability*, Vol.R-33, No.2, June 1984, pp. 176-183.

[3] A.Csenki, "Recovery Block Reliability Analysis with Failure Clustering", in *Proc. Int'l Working Conf. on Dependable Computing for Critical Applications,* Aug.1989.

[4] W.Farr, "Software Reliability Modeling Survey", in *Handbook of Software Reliability Engineering,* M.R.Lyu (Ed.), McGraw-Hill, 1996, pp. 71-117.

[5] A.L.Goel, "Software Reliability Models: Assumptions, Limitations, and Applicability" *IEEE Trans. on Software Engineering*, Vol.11, No.12, Dec.1985, pp. 1411-1423.

[6] K.Goševa-Popstojanova, A.Grnarov, "Performability and Reliability Modeling of N Version Fault Tolerant Software in Real Time Systems", in *Proc. 23rd Euromicro Conference*, 1997, pp. 532-539.

[7] K.Goševa-Popstojanova, K.Trivedi, "Failure Correlation in Software Reliability Models", in *Proc. 10th IEEE Int'l Symp. Software Reliability Engineering*, 1999, pp. 232-241. An expanded version of this paper has appeared in IEEE Trans. on Reliability, Vol.49, No.1, March 2000, pp. 37-48.

[8] K.Goševa-Popstojanova, K.Trivedi, "The Effects of Failure Correlation on Software Reliability and Performability", in *Fast Abstracts 29th Int'l Symp. Fault Tolerant Computing*, 1999, pp. 45-46.

[9] K.Goševa-Popstojanova, K.S.Trivedi, "Architecture Based Software Reliability", in *Proc. Int'l Conf. Applied Stochastic System Modeling*, 2000, pp. 41-52.

[10] D.Hamlet, "Are We Testing for True Reliability?", *IEEE Software*, July 1992, pp. 21-27.

[11] J.Laprie, K.Kanoun, "Software Reliability and System Reliability", in *Handbook of Software Reliability Engineering*, M.R.Lyu (Ed.), McGraw-Hill, 1996, pp. 27-69.

[12] J.D.Musa, A.Iannino, K.Okumoto, *Software Reliability: Measurement, Prediction, Application*, Mc Grow-Hill, 1987.

[13] A.P.Nikora, M.R.Lyu, "Software Reliability Measurement Experience", in *Handbook of Software Reliability Engineering*, M.R.Lyu (Ed.), McGraw-Hill, 1996, pp. 255-301.

[14] R.Pyke, "Markov Renewal Processes with Finitely Many States", *Annals of Mathematical Statistics*, Vol.32, Dec 1961, pp. 1243-1259.

[15] C.V.Ramamoorthy, F.B. Bastani, "Software Reliability – Status and Perspectives", *IEEE Trans. on Software Engineering*, Vol.8, No.4, July 1982, pp. 354-371.

[16] M.Sahinoglu, "Compound-Poisson Software Reliability Model" *IEEE Trans. on Software Engineering*, Vol.18, No.7, July 1992, pp. 624-630.

[17] L.A.Tomek, J.K.Muppala, K.S.Trivedi, "Modeling Correlation in Software Recovery Blocks", *IEEE Trans. on Software Engineering*, Vol.19, No.11, Nov 1993, pp. 1071-1086.

[18] M.Xie, *Software Reliability Modelling*, World Scientific Publishing Company, 1991.