Failure Correlation in Software Reliability Models

Katerina Goševa – Popstojanova and Kishor Trivedi *
Center for Advanced Computing and Communication
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708 – 0291
E-mail: {katerina, kst}@ee.duke.edu

Abstract

Perhaps the most stringent restriction that is present in most software reliability models is the assumption of independence among successive software failures. Our research was motivated by the fact that although there are practical situations in which this assumption 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 software reliability modeling framework based on Markov renewal processes which naturally introduces dependence among successive software runs. The presented approach enables the phenomena of failure clustering to be precisely characterized and its effects on software reliability to be analyzed. Furthermore, it also provides bases for a more flexible and consistent model formulation and solution. The Markov renewal model presented in this paper can be related to the existing software reliability growth models, that is, a number of them can be derived as special cases under the assumption of failure independence.

Our future research is focused on developing more specific and detailed models within this framework, as well as statistical inference procedures for performing estimations and predictions based on the experimental data.

1. Introduction

Software reliability is widely recognized as one of the most important aspects of software quality spawning a lot of research effort into developing methods of quantifying it. Despite the progress in software reliability modeling, the usage of the models is restricted by often unrealistic assumptions made to obtain mathematically tractable models and by the lack of enough experimental data. Among the basic assumptions made by various software reliability models, one which appears to be the weakest point is the independence among successive software runs.

Most existing software reliability growth models (SRGM) assume that the testing is performed homogeneously and randomly, that is, the test data are chosen from the input space by some random mechanism and the software is tested using these data assuming homogeneous conditions. In practical situations usually this is not the case. During the testing phase, different test scenarios are usually grouped according to high level functionalities which means that a series of related test runs are conducted. In addition, input data are usually chosen in order to increase the testing effectiveness, that is, to detect as many faults as possible. As a result, once a failure is observed, usually a series of related test runs are conducted to help isolate the cause of failure. Overall, testing of software systems employ a mixture of the structured (centered around scenarios), clustered (focused on fault localization) and random testing [25].

The stochastic dependence of successive software runs also depends on the extent to which internal state of a software has been affected and on the nature of operations undertaken for execution resumption (i.e., whether or not they involve state cleaning) [12].

Assuming the independence among successive software runs does not seem to be appropriate in many operational usages of software either. For instance, in many applications, such as real-time control systems, the sequence of input values to the software tend to change slowly, that is successive inputs are very close to each other. For these reasons, given a failure of a software for a 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 the software tend to occur close to each other which can result in a succession of failures.

To summarize, there may be dependencies among successive software runs, that is, the assumption of the independence of software failures could be easily violated. It means that, if a software failure occurs there would tend to

*Supported in part by the National Science Foundation, by Bellcore and by the Lord Foundation as a core project in the Center for Advanced Computing and Communication

be an increased chance that another failure will occur in the near term. We say that software failures occur in clusters if failures have tendency to occur in groups. That is, the times between successive failures are short for some periods of time and long for other periods of time.

Prevalent SRGM fall into two categories: time between failure (TBF) models which treat the inter-failure interval as a random variable, and failure count (FC) models which treat the number of failures in a given period as a random variable. In the case of TBF models the parameters of the inter – failure distribution change as testing proceeds, while the software reliability evolution in FC models is described by letting the parameters of distribution, such as mean value function, be suitable functions of time. It is worth pointing out that the two approaches presented above are strictly related. Failure time intervals description and failure counting process description are essentially two different ways of looking at the same phenomenon. To some extent, it is possible to switch between them. Herein, the analysis of the existing models and the correspondence between the two classes will not be pursued any further. For survey on the existing SRGM the reader is referred to [1], [16], [18], [20], [28].

One of the basic assumptions common to both classes of models is that the failures, when the faults are detected, are independent. For example, in [7] this assumption is included in the Standard Assumptions that apply for each presented model. In other words, neither TBF nor FC models statistically satisfy the requirements of addressing the issue of dependencies among successive software runs which usually results in failure clustering. One of the reasons "Why conventional reliability theory fails" for software, listed in [9], is that the program runs are not always independent.

To the best of our knowledge, there are only a few published papers that consider failure correlation. The empirically developed Fourier series model proposed in [6] can be used for analyzing clustered failure data, especially those with cyclic behavior. The Compound – Poisson software reliability model presented in [22] considers multiple failures that occur simultaneously in bunches within the specified CPU second or time unit. The work presented in [26] considers the problem of modeling correlation between successive executions of the software fault – tolerance technique based on recovery blocks.

In this paper we propose a software reliability modeling framework, based on Markov renewal processes, which is capable of incorporating the possible dependence among successive software runs, that is, the effect of clustering. Markov renewal model formulation has several advantages, both theoretical and practical, such as:

- *Flexible and more consistent modeling of software reliability.* The model is constructed in two stages. First, we consider the outcomes of successive software runs to construct the model in discrete time. Then, considering the execution times of the software runs we build a model in continuous time.
- *Adaptibility of the model to both dependent and independent sequences of software runs.* The model naturally introduces dependence among successive software runs, that is, failure correlation. Considering the independence among software runs is a special case of the proposed modeling framework.
- *Applicability to different phases of the software life cycle.* The proposed modeling approach is applicable for testing (debugging) phase, as well as for validation phase and operational phase.

2. Markov renewal processes - brief overview

Consider a process constructed as follows. First take a k-state 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. At the end of the interval we imagine a point event of type j. Such a process is called semi – Markov process (SMP), and it is a generalization of both continuous and discrete time Markov processes with countable state spaces. A descriptive definition of SMP would be that it is a stochastic process which moves from one state to another among a countable number of states with the successive states visited forming a discrete time Markov chain, and that the process stays in a given state a random length of time, the distribution function of which may depend on this state as well as on the one to be visited next.

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 [4], [5]. 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. MRP are of general interest since they join together the theory of two different types of processes, the renewal process and the Markov chain. In studying MRPs it is possible to follow the methods used for renewal processes, or base the treatment much more heavily on the theory of Markov chains.

In the renewal theory, the basic process studied is the number of renewals $\{N(t); t \geq 0\}$ in the interval $(0, t]$. If one regards the MRP as consisting of k dependent processes $N_1(t)$, ... , $N_k(t)$, where $N_i(t)$ refers to the points of class i, that is, the number of times state i has been visited, the observed process of points is the superposition

$N(t) = N_1(t) + \ldots + N_k(t)$. Many of the properties of Markov renewal processes are derived from those of renewal processes [4]. For example, the points of particular type form a renewal process, so that if these points are of interest, then it is necessary to consider only the distribution of an interval between successive points of this type and to make use of the standard results of renewal theory.

3. Software reliability modeling framework based on MRP

We have developed a software reliability modeling framework based on the Markov renewal process which is intuitive and naturally introduces dependence among successive software runs. MRP approach allows us to build the model in two stages. 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, that is, duration of runs, to the transitions of the DTMC. Thus, we obtain an SMP which describes both failure and execution behavior of the software. Since in software reliability theory we are interested in the distribution of the time to the next failure and the number of software failures in time interval of duration t, we focus on the equivalent point process, that is, the MRP.

3.1. Software reliability 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 that distinguishes whether the outcome of that particular run resulted in success or failure:

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

If we focus attention on failures and score 1 each time a failure occurs and 0 otherwise, then the cumulative score S_n is the number of runs that have resulted in a failure among n successive software runs. S_n can be written as the sum

$$S_n = Z_1 + \ldots + Z_n \tag{1}$$

of n possibly dependent random variables.

Suppose that if the i-th run results in failure then the probability of failure at the $(i+1)$-st run is q and the probability of success at the $(i+1)$-st run is $1-q$, that is

$$P\{Z_{i+1} = 1 | Z_i = 1\} = q$$
$$P\{Z_{i+1} = 0 | Z_i = 1\} = 1 - q.$$

Similarly, if the i-th run results in success then there are probabilities p and $1-p$ of success and failure respectively at the $(i+1)$-st run

$$P\{Z_{i+1} = 0 | Z_i = 0\} = p$$
$$P\{Z_{i+1} = 1 | Z_i = 0\} = 1 - p.$$

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

$$P = \begin{bmatrix} p & 1-p \\ 1-q & q \end{bmatrix}, \qquad 0 \leq p, q \leq 1. \tag{2}$$

Since p and q are probabilities, it follows that

$$|p + q - 1| \leq 1. \tag{3}$$

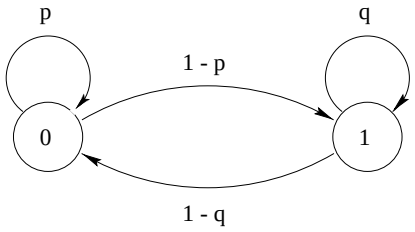

Figure 1. Markov interpretation of dependent Bernoulli trials

Let us first consider in some detail the Markov chain. The probability q (p) is the conditional probability of failure (success) on a software run given that the previous run has failed (succeeded). The unconditional probability of failure on the $(i+1)$-st run is:

$$\begin{aligned} & P\{Z_{i+1} = 1\} \\ & = P\{Z_{i+1} = 1, Z_i = 1\} + P\{Z_{i+1} = 1, Z_i = 0\} \\ & = P\{Z_{i+1} = 1 | Z_i = 1\}\, P\{Z_i = 1\} \\ & \quad + P\{Z_{i+1} = 1 | Z_i = 0\}\, P\{Z_i = 0\} \\ & = q\, P\{Z_i = 1\} + (1-p)\, P\{Z_i = 0\} \\ & = q\, P\{Z_i = 1\} + (1-p)\, [1 - P\{Z_i = 1\}] \\ & = (1-p) + (p+q-1)\, P\{Z_i = 1\}. \end{aligned} \tag{4}$$

If $p + q = 1$ the Markov chain describes a sequence of independent Bernoulli trials. In that case the equation (4) reduces to $P\{Z_{i+1} = 1\} = 1 - p = q$, which means that the failure probability does not depend on the outcome of the previous run. That is, each subsequent run has independently probabilities p and $q = 1 - p$ of being a success and

failure. This means that all software runs are independent of each other, and the number of failures S_n is a sum of n mutually independent Bernoulli random variables.

If $p + q \neq 1$ then the DTMC describes the sequence of dependent Bernoulli trials and enables us to accommodate possible dependence among successive runs. In this case the outcome of the software run (success or failure) depends on the outcome of the previous run as in equation (4). The number of failures in n runs, given by S_n, is the sum of n dependent random variables. Depending on the relation between the conditional probabilities p and q we can describe the presence or the lack of failure clustering.

When $p + q > 1$, runs are positively correlated, that is, if software failure occurs in i-th run, there would be an increased chance that another failure will occur in the next run. It is obvious that in this case failures occur in clusters. The boundary case arises when equality in (3) holds, i.e., $p + q = 2$ ($p = q = 1$). This means that if the sequence of software runs starts with failure all successive runs will fail or if it starts with success all successive runs will succeed, that is, the Markov chain remains forever in its initial state as shown in Figure 2.

Figure 2. DTMC for the case $p = q = 1$

Next consider the case when successive software runs are negatively correlated $p + q < 1$. In other words, if software failure occurs in i-th run, there would be an increased chance that a success will occur in $(i + 1)$-st run, that is, there is a lack of clustering. In the boundary case, when the equality in (3) holds ($p + q = 0$ i.e. $p = q = 0$), the Markov chain alternates deterministically between the two states, as in Figure 3.

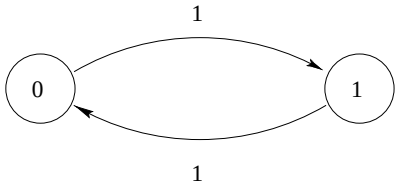

Figure 3. DTMC for the case $p = q = 0$

The boundary cases are excluded from further analysis since they are somewhat trivial, with no practical interest. We impose the condition $0 < p, q < 1$ on transition probabilities, which implies that $|p + q - 1| < 1$. In other words, DTMC in Figure 1 is irreducible and aperiodic [27].

During the testing phase software is subjected to a sequence of runs, making no changes if there is no failure. When a failure occurs on any run an attempt will be made to fix the underlying fault which will cause the conditional probabilities of success and failure on the next run to change. This implies that p and q, that is, the transition probability matrix P stays constant between two successive visits to the failure state. In other words, the transition probability matrix P_i given by

$$P_i = \begin{bmatrix} p_i & 1 - p_i \\ 1 - q_i & q_i \end{bmatrix} \tag{5}$$

defines the values of p_i and q_i for the testing runs that follow the occurrence of the i-th failure up to the occurrence of the next $(i + 1)$-st failure. Thus, the software reliability growth model in discrete time can be described with a sequence of dependent Bernoulli trials with state – dependent probabilities. The underlying stochastic process is a non – homogeneous discrete time Markov chain.

The sequence $S_n = Z_1 + \ldots + Z_n$ provides an alternative description of reliability growth model considered here. $\{S_n\}$ defines the DTMC presented in Figure 4. Both states i and i_s represent that failure state has been occupied i times. The state i represents the first trial for which the accumulated number of failures S_n equals i, while i_s represents all subsequent trials for which $S_n = i$, that is, all subsequent successful runs before the occurrence of next $(i+1)$-st failure. Without loss of generality we assume that the first run is successful, that is, 0 is the initial state.

3.2. Software reliability model in continuous time

The next step in the model construction is to obtain a process in continuous time considering the time that takes software runs to be executed by assigning a distribution function $F_{ij}(t)$ to the time spent in a transition from state i to state j of the DTMC in Figure 1. It seems realistic to assume that the runs execution times are not identically distributed for successful and failed runs. Thus, distributions $F_{ij}(t)$ depend only of the type of point at the end of the interval, that is, $F_{00}(t) = F_{10}(t) = F_{ex_S}(t)$ 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 duration of successful and failed runs, respectively. This is equivalent to assigning distribution function $F_{ex_S}(t)$ to all transitions to states i_s and $F_{ex_F}(t)$ to all transitions to states i $(i \geq 1)$ of the DTMC in Figure 4.

For the sake of simplicity, we have chosen the same execution distribution regardless of the outcome $F_{ex}(t) = F_{ex_S}(t) = F_{ex_F}(t)$. Thus, the execution time T_{ex} of each software run has the distribution function $F_{ex}(t) = P\{T_{ex} \leq t\}$. Considering the situation when software execution times are not identically distributed for successful

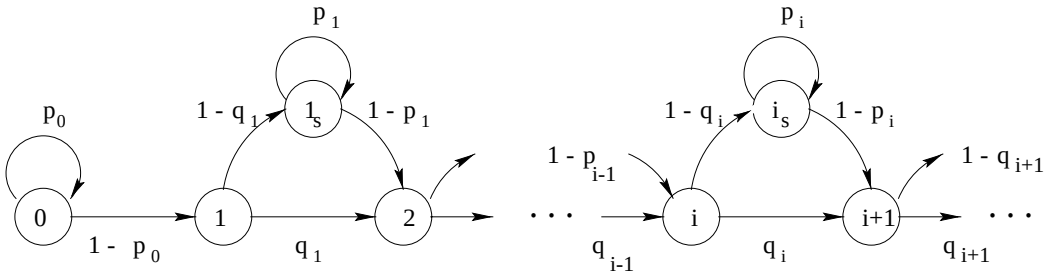

Figure 4. DTMC for testing phase

and failed runs $F_{ex_S}(t) \neq F_{ex_F}(t)$ is straightforward and will be discussed later.

With the addition of $F_{ex}(t)$ to the DTMC, we obtain the software reliability model in continuous time, that is, an MRP. 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 $(0, t]$.

In the software reliability modeling we are interested in deriving the distribution of the time between successive failures. This means that 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 and to make use of the standard results of renewal theory.

First we derive the distribution of the discrete random variable X_{i+1} defined as a number of runs between two successive visits to the failure state of the DTMC, that is, a number of runs between the i-th and $(i+1)$-st failure. Clearly, (see the Figure 4) the random variable $X_{i+1}\,(i \geq 1)$ has the pmf:

$$P\{X_{i+1} = k\} = \begin{cases} q_i & \text{if } k = 1 \\ (1-q_i)p_i^{k-2}(1-p_i) & \text{if } k \geq 2. \end{cases} \tag{6}$$

Next, we derive the distribution of the time to next failure in continuous time which is the interval distribution of the point process $N_F(t)$ that records the number of failures in $(0, t]$. Let $T_i'(i = 1, 2, \dots)$ be a continuous random variable representing the i-th failure time. Define T_{i+1} as the time elapsed from i-th failure until the occurrence of $(i+1)$-st failure. It follows that $T_{i+1}' = \sum_{j=1}^{i+1} T_j = T_i' + T_{i+1}$, where $T_0' = 0$.

When software reliability growth is considered a usual measure of reliability is conditional reliability. Since the system has experienced i failures, conditional reliability is the survivor function associated with $(i+1)$-st failure. In other words, we are concerned with the distribution of the time between successive failures $T_{i+1} = T_{i+1}' - T_i'$. It follows from (6) that the conditional distribution of the time to the next failure $F_{i+1}(t) = P\{T_{i+1} \leq t\}$ for $i \geq 1$ is given by:

$$F_{i+1}(t) = q_i F_{ex}(t) + \sum_{k=2}^{\infty} (1-q_i)\, p_i^{k-2}\, (1-p_i)\, F_{ex}^{k*}(t) \tag{7}$$

where F_{ex}^{k*} denotes the k-fold convolution of the distribution F_{ex}.

The Laplace – Stieltjes transform (LST) of $F_{ex}(t)$ is defined for $s \geq 0$ by $\tilde{F}_{ex}(s) = \int_0^\infty e^{-st} dF_{ex}(t) = E[e^{-st}]$. The LST of the distribution of the time to failure $F_{i+1}(t)$ becomes

$$\begin{aligned} \tilde{F}_{i+1}(s) &= q_i \tilde{F}_{ex}(s) + \sum_{k=2}^{\infty} (1-q_i)\, p_i^{k-2}\, (1-p_i)\, \tilde{F}_{ex}^{k}(s) \\ &= \frac{q_i \tilde{F}_{ex}(s) + (1-p_i-q_i)\, \tilde{F}_{ex}^2(s)}{1 - p_i \tilde{F}_{ex}(s)}. \end{aligned} \tag{8}$$

The inversion of (8) is in principle straightforward and reasonably simple closed – form results can be obtained when the run's execution time distribution $F_{ex}(t)$ has rational LST (such as phase – type distributions discussed in [5]).

We can develop some general properties of time to next failure without making assumptions about the form of the run's execution time distribution $F_{ex}(t)$. Due to the well known property of LST the moments can be derived easily by a simple differentiation of (8)

$$\begin{aligned} E[T_{i+1}] &= -\left.\frac{d\tilde{F}_{i+1}(s)}{ds}\right|_{s=0} = -\frac{2-p_i-q_i}{1-p_i}\left.\frac{d\tilde{F}_{ex}(s)}{ds}\right|_{s=0} \\ &= \frac{2-p_i-q_i}{1-p_i} E[T_{ex}] \end{aligned} \tag{9}$$

where $E[T_{ex}]$ is the run's mean execution time. If $p_i + q_i = 1$ then successive runs are independent and (9) becomes

$$E[T_{i+1}^{in}] = \frac{E[T_{ex}]}{1-p_i}$$

which means that we can rewrite (9) as

$$E[T_{i+1}] = (2 - p_i - q_i)\ E[T_{i+1}^{in}].$$

If we introduce the notation $\pi_i = p_i + q_i - 1$ then

$$E[T_{i+1}] = (1 - \pi_i)\ E[T_{i+1}^{in}]. \tag{10}$$

When the successive software runs are dependent ($\pi_i \neq 0$) two cases can be considered:

1. If $0 < \pi_i < 1 \quad (p_i + q_i > 1)$ then successive runs are positively correlated, that is, the failures occur in clusters. It follows that the mean time between failures is shorter than it would be if the runs were independent. In other words, SRGM that assume independence among failures will result in overly optimistic predictions.

2. If $-1 < \pi_i < 0 \quad (p_i + q_i < 1)$ then the successive runs are negatively correlated and the the mean time between failures is longer then it would be if the runs were independent.

3.3. Model generalizations

The presented model can be generalized in many ways. First, considering not identically distributed runs execution times for successful and failed runs $F_{ex_S}(t) \neq F_{ex_F}(t)$ is straightforward: we assign distribution function $F_{ex_S}(t)$ to each transitions to state 0 (successful runs), and $F_{ex_F}(t)$ to each transition to state 1 (failed runs). By making appropriate changes in (7) we get:

$$F_{i+1}(t) = q_i F_{ex_F}(t) + \sum_{k=2}^{\infty} (1 - q_i)\ p_i^{k-2}\ F_{ex_S}^{(k-1)*}(t)\ (1 - p_i)\ F_{ex_F}(t) \tag{11}$$

which leads to LST transform:

$$\tilde{F}_{i+1}(s) = \frac{q_i \tilde{F}_{ex_F}(s) + (1 - p_i - q_i)\ \tilde{F}_{ex_F}(s)\ \tilde{F}_{ex_S}(s)}{1 - p_i \tilde{F}_{ex_S}(s)}. \tag{12}$$

Next generalization considers the possible variability of the run's execution time distribution $F_{ex}(t)$. We have assumed that this distribution does not change during the whole testing phase. This assumption can be violated for reasons such as significant changes in the code due to the fault fixing attempts. In other words, it may be useful to consider the situation when the parameters of the distribution $F_{ex}(t)$ change after each failure occurrence (i.e., after each visit to the failure state). In that case, we need only to substitute $F_{ex}(t)$ with $F_{ex_i}(t)$ in equation (7).

One more generalization that can be considered in applying the MRP approach is to choose k-state Markov chain to describe the sequence of software runs in discrete space. For example, it is possible to define DTMC which considers more than one type of failure. We can also add states that will enable us to consider non – zero time to remove a fault or periods of time when the software is idle.

3.4. Applicability to validation phase and operational phase

The presented modeling approach can be used to make reliability estimations of the software based on the results of testing performed in its validation phase. In the case of safety critical software it is common to demand that all known faults are removed. This means that if there is a failure during the testing the fault must be identified and removed. After the testing (debugging) phase the software enters a validation phase in order to show that it has a high reliability prior to actual use. In this phase no changes are made to the software. Usually, the validation testing for safety critical software takes the form of specified number of test cases or specified period of working that must be executed failure – free. While of particular interest in the case of safety critical software, the same approach can be applied to any software that, in its current version, has not failed for n successive runs.

Let θ be the unconditional probability of a failure per run (execution). For the case of independent Bernoulli trials the probabilities that each run will fail $\theta = q$ or succeed $1 - \theta = p$ are independent of previous runs. Thus, the probability that n independent test runs are conducted without failure is

$$(1 - \theta)^n = (1 - q)^n = p^n. \tag{13}$$

The largest value of θ such that

$$(1 - \theta)^n \geq \alpha \tag{14}$$

defines the $1 - \alpha$ upper confidence bound on θ [9], [27]. Solving (14) for θ gives $1 - \alpha$ confidence that the failure probability per execution of a software that reveals no failure in n independent runs is below

$$\theta^* = 1 - \alpha^{1/n}. \tag{15}$$

For the related recent work on a Bayesian estimation approach see [15] and [17].

Now consider a sequence of possibly dependent software runs. During the validation phase the software is not changing, that is, p and q do not vary. In other words, the sequence of runs can be described by the homogeneous DTMC with transition probability matrix (2). We assume that the DTMC is stationary, that is, each run has the same probability of

failure $P\{Z_{i+1}=1\}=P\{Z_i=1\}=\theta$. Then, from (4) it follows that

$$\theta=\frac{1-p}{2-p-q}. \tag{16}$$

The probability that n successive runs will succeed is given by:

$$\begin{aligned}&P\{Z_n=\ldots=Z_2=Z_1=0|Z_0=1\}\\&=P\{Z_n=0|Z_{n-1}=0\}\ldots P\{Z_2=0|Z_1=0\}\\&\quad\times P\{Z_1=0|Z_0=1\}\\&=p^{n-1}\ (1-q).\end{aligned} \tag{17}$$

Using the notion $\pi=p+q-1$ in addition to (16) we can rewrite (17) in terms of θ and π. It follows that in the case of dependent Bernoulli trials $(1-\alpha)$ upper confidence bound on failure probability per execution θ becomes:

$$\theta^*=\frac{1-\alpha^{1/(n-1)}}{1-\pi}. \tag{18}$$

If failures occur in clusters ($0<\pi<1$) then the confidence bound on failure probability is approximately $1/(1-\pi)$ higher than the bound obtained under the independence assumption (15), that is, the result obtained under the assumption of independence is overly optimistic.

We consider the applicability of the Markov renewal model in the operational phase next. During the operational phase no changes are made to the software, that is, the sequence of runs can be described by the homogeneous DTMC. The well developed theory of Markov renewal processes [5], [21] and its limiting results can be used to derive a number of measures other than the distribution of the time between failures, such as

- the expected number of failures $M_F(t)=E[N_F(t)]$ and the expected number of successes $M_S(t)=E[N_S(t)]$ in the interval $(0,t]$;
- the probability of success at time t, that is, instantaneous availability;
- the limiting probability that the process will be in state 0 at time t as $t\to\infty$, that is, steady – state availability.

4. Some special cases and relation to existing models

The Markov renewal model described in this paper can be related to existing software reliability models. The model in discrete time is comparable with an input – domain based models which are static and don't consider the time, while the model in continuous time is comparable with time – domain software reliability growth models. We next examine these relations.

The software reliability model in discrete time presented in this paper can be related to input – domain based models which consider the software input space from which test cases are chosen. For example, the input – domain model presented in [19], [20] defines the conditional reliability as probability that k runs will succeed before the failure, given that j faults have been detected and corrected

$$R_j(k|\lambda_j)=(1-\lambda_j)^k\ \lambda_j \tag{19}$$

where $k\geq 0$ and $\lambda_j\quad(0\leq\lambda_j\leq 1)$ is defined as a fault size under operational inputs. It is obvious that (19) is a special case of (6) under the assumption that successive inputs have independent failure probability. In [19] λ_j is treated as a random variable with a distribution function $G(\lambda_j)$, and (19) becomes $R_j(k)=\int(1-\lambda_j)^k\,\lambda_j dG(\lambda_j)=E[(1-\lambda_j)^k\ \lambda_j]$.

Note that the model in discrete time is also related to the Compound – Poisson software reliability model in [22]. This model is based on the work presented in [23] which approximates the sum of dependent integer – valued random variables S_n given by (1) with a compound Poisson process.

Now consider the software reliability model in continuous time. If $\tilde{F}_{ex}(s)$ is a rational function of s, so too is $\tilde{F}_{i+1}(s)$, and the inversion of (8) is in principle straightforward. An important special case is 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 time – domain based software reliability models and some of their basic assumptions, such as the independence of successive runs and the exponential distribution of the inter – failure times. In other words, the simplest special case of the model is under the assumptions that the successive software runs are independent ($p_i+q_i=1$) and the software execution time (duration of testing runs) is exponentially distributed with rate μ. Inverting (8) leads to the pdf of the inter – failure times:

$$f_{i+1}(t)=(1-p_i)\ \mu\, e^{-(1-p_i)\mu\,t}, \tag{20}$$

that is, the conditional reliability is given by

$$R_{i+1}(t)=1-F_{i+1}(t)=e^{-(1-p_i)\mu\,t}. \tag{21}$$

It follows that the inter – failure time is exponentially distributed with rate $(1-p_i)\,\mu$ if the software testing is considered as a sequence of independent runs with an exponential distribution of the execution times. Note that, the coefficient of variation, defined by the ratio of standard deviation and mean, of an exponential distribution is 1.

The alternative interpretation of this special case can be found in [11]. Under the assumption that inputs arrive at software system according to a Poisson process with rate μ

which is interpreted as intensity of testing, the probability that the software encounters no failures in a time interval $(0, t]$ is given by:

$$1 - F(t) = \sum_{j=0}^{\infty} \frac{e^{-\mu t}(\mu t)^j}{j!} \left(\frac{M - M^*}{M}\right)^j \qquad (22)$$

where M is the size of the input data space, i.e., the number of data items which can be used as input to the software and M^* is the total number of these input data which may cause software failure. The first term inside the summation sign denotes the probability that j inputs are received in time t, and the second term denotes that none of j inputs lead to a failure of software. It is easy to verify that (22), when simplified, leads to

$$1 - F(t) = e^{-\frac{M^*}{M}\mu t}. \qquad (23)$$

It means that the time to first failure has an exponential distribution with failure rate $\lambda = (M^*/M)\mu$. Note that M^*/M is just the probability of failure on a test run.

The conditional reliability, that is, the survivor function of the time between $(i-1)$-st and i-th failure T_i becomes

$$1 - F_i(t) = P\{T_i > t\} = e^{-\lambda_i t}. \qquad (24)$$

Even though the motivations are different and the parameters have different interpretations, mathematically the model derived in [11] is a special case of the software reliability model based on MRP under the assumptions that successive software runs (i.e., failures) are independent and the software execution times are exponentially distributed with rate μ. In [11] it is shown that the Jelinski – Moranda model [10] can be obtained by introducing $\lambda_i = \frac{\mu}{M}(N - i + 1) = \Lambda(N - i + 1)$ and treating Λ and N as unknown parameters. Then by adopting a Bayesian point of view two other models can be derived from the Jelinski – Moranda model [11]:

1. Goel – Okumoto model [8] is obtained by assuming Λ has a known value and assigning Poisson prior distribution for N;

2. Littlewood – Verrall model [14] is obtained by assuming N has a known value and that Λ has a prior Gamma distribution.

Some other time – domain SRGM can easily be obtained as special cases under the assumption of independence. Due to the space limitations and vast number of SRGM the analysis of the relation to the existing models will not be pursued any further.

Let us now keep the assumption that the distribution of the run's execution time is exponential, but assume dependence between successive software runs ($p_i + q_i \neq 1$). Inverting (8) leads to the pdf of the inter – failure time given by:

$$f_{i+1}(t) = \frac{(1 - q_i)}{p_i}(1 - p_i)\mu e^{-(1-p_i)\mu t} + \frac{(p_i + q_i - 1)}{p_i}\mu e^{-\mu t}. \qquad (25)$$

This distribution is a mixture (compound) distribution [27] with pdf of a form $g(t) = \alpha_1\, g_1(t) + \alpha_2\, g_2(t)$, where $\alpha_1, \alpha_2 > 0$ and $\alpha_1 + \alpha_2 = 1$.

In the case when $p_i + q_i > 1$ the inter-failure distribution (25) is hyperexponential, that is, a mixture of two exponential distributions with rates $(1 - p_i)\,\mu$ and μ. The mixing probabilities are given by $\alpha_1 = (1 - q_i)/p_i$ and $\alpha_2 = (p_i + q_i - 1)/p_i$, respectively. Note that the coefficient of variation in the case of hyperexponential distribution is greater than 1. It is obvious that due to the presence of failure clustering, the inter – failure time has smaller mean $E[T_{i+1}] < E[T^{in}_{i+1}]$ and greater variability compared to the independent case, even under the assumption of exponentially distributed duration of testing runs.

If the $p_i + q_i < 1$ then (25) becomes a mixture of an exponential distribution with rate $(1 - p_i)\,\mu$ and hypoexponential distribution with rates $(1 - p_i)\,\mu$ and μ. The mixing proportions are $\alpha_1 = q_i/(1 - p_i)$ and $\alpha_2 = (1 - p_i - q_i)/(1 - p_i)$, respectively. It can be shown that the coefficient of variation in this case is smaller than 1. In other words, the inter – failure time has greater mean $E[T_{i+1}] > E[T^{in}_{i+1}]$ and smaller variability compared to the independent case.

The presented results clearly demonstrate the effects of failure correlation on the software reliability measures. It is obvious that some of the common assumptions made by software reliability models are inadequate and result in optimistic estimations when failures are indeed clustered.

5. Discussion and future work

The ultimate goal when developing reliability growth models is the development of good reliability inference and prediction methods which can be applied to software development. This paper does not deal with inference or predictions per se. It is mainly aimed at showing that the classical software reliability theory can be extended in order to consider a sequence of possibly dependent software runs, that is, a failure correlation. However, there are many research issues that we would like to address in near future in order the model to be fully specified and applied in real software development projects for performing estimations and predictions.

Let us consider in some detail the concept of software runs. The operation of software can be broken down into series of runs [18]. Each run performs mapping between a set of input variables and a set of output variables and

consumes a certain amount of execution time. The input variable for a program run is any data item that exists external to the run and is used by a run. There does not have to be a physical input process. The input variable may simply be located in memory, waiting to be accessed. 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 [18]. The information associated with software runs can generally be grouped into two categories:

- *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.

- *Input and Outcome.* The descriptive information about each specific run generally specifies the input for the program, the testing environment, and the outcome that has been obtained (success or failure).

Relevant data about the failed runs, as well as for successful runs are routinely captured in many projects for test tracking and testing process management purposes [24], [25]. This data, with possibly minor modifications, provide common source for reliability growth modeling, input – domain analysis, and integrated analysis in this paper.

The existing time – domain SRGM disregard the successful runs between two failures and ignore the information conveyed by them. The consideration of successful runs, that is, non – failure stops of software operation for parameters estimation of some of the existing SRGM was presented recently in [3]. In this work it is pointed out that disregarding the non – failure stops violates the Maximum Information Principle that suggests to exploit available software reliability information as much as possible.

In the case of MRP model the time between failures T_{i+1} is a random variable whose distribution depends on the distribution of the run's execution time $F_{ex}(t)$, and on the conditional probabilities p_i and q_i. The timing information associated with each run can be obtained quite easily in many computer systems. Therefore, instead of making assumptions, the specific distribution function of the run's execution time $F_{ex}(t)$ could be determinated from measured data.

Consider possible models for the parameter set $\{p_1, q_1, p_2, q_2, \dots\}$ next. It is mathematically possible to permit an infinite number of failures, and in this case the parameter set is infinite. By setting $q_i = 0$ and $p_i = 1$ for $i \geq n + 1$ the finite failure model is obtained as a special case.

For the model to be fully, specified it is necessary to consider the way the parameters change as a result of the fault removal attempts. In this regard, there are two possible approaches. One approach of modeling the parameter set is to relate them to the number of test runs or to the number of faults in a systematic way by assuming various deterministic functional relationships. That is, the free parameters are simply seen as unknown constants to be estimated by the sample data. The alternate approach is to treat the parameters as random variables themselves in order to consider the uncertainty about the effect of fault removal [13], [14]. Thus, even if the fault removal is successful, there will be uncertainty about the size of the fault removed, and so uncertainty about the magnitude of reliability improvement. This approach results in a doubly stochastic model: one model for set of parameters and the other for the times between failures conditional on the parameters.

In SRGM it is generally assumed that when debugging is taking place there is a presence of reliability growth. However, the detailed assumptions about the nature of this growth and the way the modeling parameters change as a result of the fault removal attempts vary from one model to another. Here, some basic issues that should be taken into account are briefly outlined [2], [28]:

- *Size of faults.* In general, different software faults do not affect equally the failure probability. Some faults which are more likely to occur contribute more to the failure probability than other faults.

- *Imperfect debugging.* Often, fault fixing cannot be seen as a deterministic process, leading with certainty to the removal of the fault. Moreover, it may cause the introduction of new faults in code.

- *Non – instantaneous and delayed fault removal.* Usually, neither the removal of a fault occurs immediately after the failure is observed, nor the time to remove the fault is negligible.

- *Changing testing strategy.* In practice it is important to deal with the case of nonhomogeneous testing, that is the model should include the possibility of describing variations of the testing strategy with time.

6. Conclusion

The research work presented in this paper is devoted to development of the more realistic software reliability modeling framework that enables to account for the phenomena of failure correlation and to study its effects on the software reliability measures. The important property of the developed Markov renewal modeling approach is its flexibility. It allows us to construct the software reliability model in both discrete time and continuous time, and depending on our goals to base the analysis either on Markov chain theory or on renewal process theory. The model presented in this paper can be related to the existing software reliability models. In fact, it is demonstrated that a number of input –

domain and time – domain models can be derived as special cases under the assumption of failure independence.

This paper is mainly aimed at showing that the classical software reliability theory can be extended in order to consider a sequence of possibly dependent software runs, that is, a failure correlation. It does not deal with inference or predictions per se. In order the model to be fully specified and applied for performing estimations and predictions in real software development projects we need to address many research issues, such as the detailed assumptions about the nature of the reliability growth and the way the modeling parameters change as a result of the fault removal attempts.

To summarize, the software reliability modeling framework presented in this paper contributes toward more realistic modeling of software reliability since it naturally integrates the phenomena of failure correlation. In addition, it also provides bases for a more flexible and consistent approach to the mathematical formulation on software reliability. However, in order to apply the model to real data the development of more detailed and specific models within this framework, as well as statistical inference procedures for modeling parameters are the subjects of our future research.

References

[1] S. Bittanti, editor. *Software Reliability Modelling and Identification*, volume 341 of *Lecture Notes in Computer Science*. Springer – Verlag, 1988.

[2] S. Bittanti, P. Bolzern, and R. Scattolini. An introduction to software reliability modeling. In S. Bittanti, editor, *Software Reliability Modelling and Identification*, volume 341 of *Lecture Notes in Computer Science*, pages 43–66. Springer – Verlag, 1988.

[3] K. Cai. Censored software – reliability models. *IEEE Transactions on Reliability*, 46(1):69–75, March 1997.

[4] D. R. Cox and V. Isham. *Point Processes*. Chapman and Hall, 1980.

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

[6] L. H. Crow and N. D. Singpurwalla. An empirically developed Fourier series model for describing software failures. *IEEE Transactions on Reliability*, R-33(2):176–183, June 1984.

[7] W. Farr. Software reliability modeling survey. In M. R. Lyu, editor, *Handbook of Software Reliability Engineering*, pages 71–117. Mc Graw – Hill, 1996.

[8] A. L. Goel and K. Okumoto. Time dependent error – detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28:206– 211, 1979.

[9] D. Hamlet. Are we testing for true reliability? *IEEE Software*, pages 21– 27, July 1992.

[10] Z. Jelinski and P. B. Moranda. Software reliability research. In W. Freiberger, editor, *Statistical Computer Performance Evaluation*, pages 485–502. Academic Press, 1972.

[11] N. Langberg and N. D. Singpurwalla. A unification of some software reliability models. *SIAM J. Sci. Stat. Comput.*, 6(3):781–790, July 1985.

[12] J. Laprie and K. Kanoun. Software reliability and system reliability. In M. R. Lyu, editor, *Handbook of Software Reliability Engineering*, pages 27–69. Mc Graw – Hill, 1996.

[13] B. Littlewood. Modeling growth in software reliability. In P. Rook, editor, *Software Reliability Handbook*, pages 137–153. Elsevier Applied Science, 1990.

[14] B. Littlewood and J. L. Verrall. A Bayesian reliability growth model for computer software. In *Proceedings of the IEEE Symposium on Computer Software Reliability*, pages 70–77, 1973.

[15] B. Littlewood and D. Wright. Some conservative stopping rules for the operational testing of safety – critical software. *IEEE Transaction on Software Engineering*, 23(11):673–683, November 1997.

[16] M. R. Lyu, editor. *Handbook of Software Reliability Engineering*. Mc Graw – Hill, 1996.

[17] K. W. Miller, L. J. Morell, R. E. Noonan, S. K. Park, D. M. Nicol, B. W. Murrill, and J. M. Voas. Estimating the probability of failure when testing reveals no failures. *IEEE Transaction on Software Engineering*, 18(1):33–43, January 1992.

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

[19] C. V. Ramamoorthy and F. B. Bastani. Modeling of the software reliability growth process. In *Proceedings of COMPSAC*, pages 161–169, 1980.

[20] C. V. Ramamoorthy and F. B. Bastani. Software reliability – status and perspectives. *IEEE Transaction on Software Engineering*, 8(4):354–371, July 1982.

[21] S. M. Ross. *Applied Probability Models with Optimization Applications*. Holden – Day, 1970.

[22] M. Sahinoglu. Compound – Poisson software reliability model. *IEEE Transaction on Software Engineering*, 18(7):624–630, July 1992.

[23] R. F. Serfozo. Compound Poisson approximations for sums of random variables. *Annals of Probability*, 14(4):1391–1398, 1986.

[24] J. Tian. Integrating time domain and input domain analyses of software reliability using tree – based models. *IEEE Transactions on Software Engineering*, 21(12):945–958, December 1995.

[25] J. Tian and J. Palma. Data partition based reliability modeling. In *Proceedings of the 7th IEEE International Symposium on Software Reliability Engineering*, pages 354–363, 1996.

[26] L. A. Tomek, J. K. Muppala, and K. S. Trivedi. Modeling correlation in software recovery blocks. *IEEE Transactions on Software Engineering*, 19(11):1071–1086, November 1993.

[27] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice – Hall, 1982.

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