Failure Correlation in Software Reliability Models

Katerina Goševa-Popstojanova, *Member, IEEE,* and Kishor S. Trivedi, *Fellow, IEEE*

***Abstract*—Perhaps the most stringent restriction in most software reliability models is the assumption of statistical 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.**

The research work in this paper is devoted to developing the software reliability modeling framework that can consider 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 construction of the software reliability model in both discrete time and continuous time, and (depending on the goals) to base the analysis either on Markov chain theory or on renewal process theory. Thus, our modeling approach is an important step toward more consistent and realistic modeling of software reliability. It can be related to existing software reliability growth models. Many input-domain and time-domain models can be derived as special cases under the assumption of failure s-independence.

This paper aims at showing that the classical software reliability theory can be extended to consider a sequence of possibly s-dependent software runs, *viz*, failure correlation. It does not deal with inference nor with predictions, *per se.* For the model to be fully specified and applied to estimations and predictions in real software development projects, we need to address many research issues, e.g., the

- **detailed assumptions about the nature of the overall reliability growth,**
- **way modeling-parameters change as a result of the fault-removal attempts.**

***Index Terms*—Failure correlation, Markov renewal process, sequence of dependent software runs, software reliability.**

I. Introduction

Acronyms[1]

SRM	software reliability model
SRGM	software reliability growth model
TBF	time between failures
FC	failure count
NHPP	nonhomogeneous Poisson process
DTMC	discrete-time Markov chain
SMP	semi-Markov process
MRP	Markov renewal process
LST	Laplace–Stieltjes transform
CB	s-confidence bound
r.v.	random variable

Manuscript received August 31, 1998; revised October 1, 1999. This research was supported in part by the US National Science Foundation under an IUCRC TIE project between Purdue SERC and CACC-Duke Grant EEC-9714965, Bellcore (now Telcordia), U.S. Department of Defense Grant EEC-9418765, and the Lord Foundation, as a core project in the Center for Advanced Computing and Communication at Duke University.

The authors are with the Department of Electrical and Computer Engineering; Duke University; Durham, NC 27708-0291 USA (e-mail: {katerina, kst}@ee.duke.edu).

Responsible editor: M.A. Vouk.

Publisher Item Identifier S 0018-9529(00)06201-1.

[1]The singular and plural of an acronym are always spelled the same

Software reliability is widely recognized as one of the most important aspects of software quality, and it spawns much research effort into developing methods of quantifying it. Despite the progress in software-reliability modeling, the use of the models is restricted by

- often unrealistic assumptions made to obtain mathematically tractable models,
- the lack of enough experimental data.

Among the basic assumptions made by various SRM, one which appears to be the weakest point is the s-independence among successive software runs.

Most existing SRGM assume that the testing is performed homogeneously and randomly, i.e., 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 this is usually not true. During the testing phase, various 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, i.e., 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 uses a mixture of [32]

- structured (centered around scenarios),
- clustered (focused on fault localization),
- random testing.

The s-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 [16].

Assuming the s-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 tends to change slowly: 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.

In summary, there can be s-dependencies among successive software runs, i.e., the assumption of the s-independence of software failures could be easily violated. Thus if a software failure occurs, there would be an increased chance that another failure will occur in the near term. Software failures occur in clusters if failures have tendency to occur in groups, i.e., 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:

- TBF models which treat the inter-failure interval as a r.v.,
- FC models which treat the number of failures in a given period as a r.v.

For 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. The two approaches here 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. In this paper, the analysis of the existing models and the correspondence between the two classes is not pursued any further. For surveys on existing SRGM see [13], [23], [25], [29], [35].

One of the basic assumptions common to both classes of models is that the failures, when the faults are detected, are s-independent. For example, in [6] 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 s-dependencies among successive software runs which usually result in failure clustering. One of the reasons "Why conventional reliability theory fails" for software [11] is that the program runs are not always s-independent.

To the best of our knowledge, there are only a few published papers that consider failure s-correlation. The empirically developed Fourier series model [5] can be used for analyzing clustered failure data, especially those with cyclic behavior. The Compound-Poisson SRM [27] considers multiple failures that occur simultaneously in bunches within the specified CPU second or time unit. The work [33] considers the problem of modeling correlation between successive executions of the software fault-tolerance technique based on recovery blocks.

This paper proposes a software reliability modeling framework, based on Markov renewal processes, which can incorporate the possible s-dependence among successive software runs, *viz,* 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 2 stages: 1) Consider the outcomes of successive software runs to construct the model in discrete time. 2) Considering the execution times of the software runs, we build a model in continuous time.
- *Adaptability of the model to both s-dependent and s-independent sequences of software runs:* The model introduces s-dependence among successive software runs, *viz,* failure s-correlation. Considering the s-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 applies to the testing (debugging) phase, as well as to the validation phase and operational phase.

Section II presents a brief overview of Markov renewal processes.

Section III presents a software reliability modeling framework based on MRP which is then used to derive SRGM that incorporates the phenomena of failure correlation and enables us to study its effects on software reliability. Some ideas about the way particular assumptions can be removed or relaxed are introduced.

Section IV presents the application of the model to the validation and operational phases.

Section V explores the relation of the MRP approach to the existing SRGM.

Section VI discusses the issues that should be considered when developing specific models within this framework.

Notation

j	ordinal number of software run
Z_j	nonindependent Bernoulli r.v.; $0 \equiv$ success, $1 \equiv$ failure
p	$\Pr\{Z_{j+1} = 0 \mid Z_j = 0\}$
$\overline{p}$	$1 - p$
q	$\Pr\{Z_{j+1} = 1 \mid Z_j = 1\}$
$\overline{q}$	$1 - q$
S_n	sum of s-dependent Z_j over $j = 1, \cdots, n$
$F_{k,l}(t)$	Cdf of time spent in a transition from state k to state l of SMP
$F_{\mathrm{ex}_S}(t)$	Cdf of the successful run's execution time
$F_{\mathrm{ex}_F}(t)$v	Cdf of the failed run's execution time
$F_{\mathrm{ex}}(t)$	Cdf of the run's execution time when $F_{\mathrm{ex}_S}(t) = F_{\mathrm{ex}_F}(t)$
$N(t)$	number of runs in $(0, t]$
$N_F(t)$	number of failed runs in $(0, t]$
$N_S(t)$	number of successful runs in $(0, t]$
i	number of failures experienced
p_i, q_i	conditional probabilities of [success, failure] for testing runs that follow the occurrence of failure i
$\overline{p}_i$	$1 - p_i$
$\overline{q}_i$	$1 - q_i$
X_{i+1}	number of runs between failures i and $(i+1)$
T'_i	occurrence time of failure i; $T'_0 = 0$
T_{i+1}	$T'_{i+1} - T'_i$: inter-failure time
t_{i+1}	realization of T_{i+1}
$R_{i+1}(t)$	$\Pr\{T_{i+1} > t \mid t_1, \cdots, t_i\}$: conditional Sf
$F_{i+1}(t)$	$1 - R_{i+1}(t)$: conditional Cdf of T_{i+1}
$f_{i+1}(t)$	conditional pdf of T_{i+1}
θ	unconditional probability of failure per run (validation and operational phase)
π_i	$p_i + q_i - 1$

Other, standard notation is given in the "Information for Readers and Authors" at the rear of each issue.

II. Markov Renewal Processes Brief Overview

Consider a process constructed as follows:

1) Take an m-state DTMC with transition probability matrix $P = [p_{k,l}]$.

2) Construct a process in continuous time by making the time spent in a transition from state k to state l have Cdf $F_{k,l}(t)$, such that times are mutually s-independent.

This is an SMP, and is a generalization of both continuous and discrete time Markov processes with countable state spaces. A descriptive definition of SMP is 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 Cdf of which can depend on this state as well as on the one to be visited next.

The family of stochastic processes used in this paper, MRP, is equivalent to the family of SMP [3], [4]. 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 MRP one can follow the methods used for renewal processes, or can base the treatment much more heavily on the theory of Markov chains.

In renewal theory, the basic process is the number of renewals $\{N(t);\, t \geq 0\}$ in the interval $(0,\, t]$. If one regards the MRP as consisting of m s-dependent processes $N_1(t), \cdots, N_m(t)$, where $N_k(t)$ refers to the points of class k, *viz,* number of times state k has been visited, then the observed process of points is the superposition $N(t) = N_1(t) + \cdots + N_m(t)$. Many of the properties of MRP are derived from those of renewal processes [3]. 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 use the standard results of renewal theory.

III. Software Reliability Modeling Framework Based on MRP

We have developed a software reliability modeling framework based on MRP which is intuitive and easily introduces s-dependence among successive software runs. Since each software run has 2 possible outcomes (success or failure), the usual way of looking at the sequence of software runs is to consider it as a sequence of s-independent Bernoulli trials, where each trial has success-probability p and failure-probability $q = 1 - p$. The two Cdf, binomial and geometric, are connected with the s-independent Bernoulli trials:

- the number of runs that have failed among n successive software runs defines a r.v. with a binomial pmf;
- the number of software runs between two failures has the geometric pmf.

Since the number of software runs n is large and q is small, the well known limiting results for these pmf are usually used as basic assumptions in SRM [18]:

- the number of failures in the limit has the Poisson distribution,
- the time to failure in the limit is exponentially distributed.

Regarding the validity of the underling assumptions, there are two questions:

1) Is the probability of failure q constant?
2) Are the runs s-independent?

Question #1 is addressed by SRGM which consider the testing phase of the software life cycle. When a failure occurs in any run, removing the underlying faults will cause the probability of failure to change. The work in this paper addresses question #2. Our goal is to show that the classical software reliability theory can be extended to consider a sequence of possibly s-dependent software runs.

The sequence of successive software runs (successful or failed) can be viewed as a realization of point events in time: a point process. Poisson process is the simplest point process. It considers only the failure points, i.e.,

- disregards the successful runs between 2 failures,
- ignores the information conveyed by them.

This paper views the sequence of s-dependent software runs, when the outcome of each run depends on the outcome of the previous run, as a sequence of s-dependent Bernoulli trials. Therefore, we consider both failed and successful runs: both failure and nonfailure stops of software operation. A convenient way of specifying a point process with more then 1 class of points is a Markov renewal process. It enable intuitive and separate discussion of the 2 elements of randomness in software operation:

- uncertainty about the outcome of the software run,
- uncertainty about the time for the run to be executed.

Thus, we build the model in 2 stages:

1) Define a DTMC which considers the outcomes from the sequence of possibly s-dependent software runs in discrete time.
2) Construct the process in continuous time by attaching the distributions of the runs execution time to the transitions of the DTMC.

The SMP describes both failure and execution behavior of the software. Since software reliability theory considers the distribution of the time to the next failure and the number of software failures in time interval of duration t, this paper focuses on the equivalent point process: the MRP

Assumptions:

1) The probability of success or failure at each run depends on the outcome of the previous run.
2) A sequence of software runs is defined as a sequence of s-dependent Bernoulli trials.
3) Each software run takes a random amount of time to be executed.
4) Software execution times are not identically distributed for successful and failed runs. ◀

A. SRM in Discrete Time

We view the sequence of software runs in discrete time as a sequence of s-dependent Bernoulli trials in which the probability of success or failure at each trial depends on the outcome of the previous trial. Associate with the software-run j a r.v. Z_j

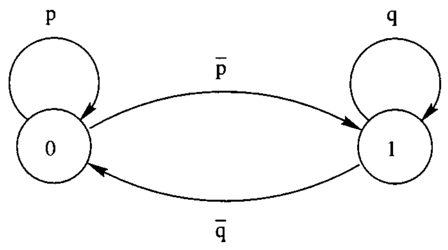

Fig. 1. Markov Interpretation of s-Dependent Bernoulli Trials

that distinguishes whether the outcome of that particular run resulted in success or failure:

$$Z_j = \begin{cases} 0 & \text{a success on run } j \\ 1 & \text{a failure on run } j. \end{cases}$$

If we focus attention on failures and score 1 each time a failure occurs and 0 otherwise, then the number of runs that have resulted in a failure among n successive software runs is:

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

of n possibly s-dependent r.v.

Suppose that if run j results in failure then, at run $j+1$, the failure probability is q and the success probability is $\bar{q}$:

$$\Pr\{Z_{j+1} = 1 | Z_j = 1\} = q,$$
$$\Pr\{Z_{j+1} = 0 | Z_j = 1\} = \bar{q}.$$

Similarly, if run j results in success then p and $\bar{p}$ are the probabilities of success and failure, respectively, at run $(j+1)$:

$$\Pr\{Z_{j+1} = 0 | Z_j = 0\} = p,$$
$$\Pr\{Z_{j+1} = 1 | Z_j = 0\} = \bar{p}.$$

The sequence of s-dependent Bernoulli trials $\{Z_j;\, j \geq 1\}$ defines a discrete-time Markov chain with 2 states. One of the states, denoted by 0, is a success; the other denoted by 1 is a failure. A graphic description of this Markov chain is provided by its state diagram in Fig. 1. Its transition probability matrix is:

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

Since p and q are probabilities, it follows that

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

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 run $(j+1)$ is:

$$\begin{aligned} &\Pr\{Z_{j+1} = 1\} \\ &\quad = \Pr\{Z_{j+1} = 1,\, Z_j = 1\} + \Pr\{Z_{j+1} = 1,\, Z_j = 0\} \\ &\quad = \Pr\{Z_{j+1} = 1 | Z_j = 1\} \cdot \Pr\{Z_j = 1\} \\ &\quad\quad + \Pr\{Z_{j+1} = 1 | Z_j = 0\} \cdot \Pr\{Z_j = 0\} \\ &\quad = q \cdot \Pr\{Z_j = 1\} + \bar{p} \cdot \Pr\{Z_j = 0\} \\ &\quad = q \cdot \Pr\{Z_j = 1\} + \bar{p} \cdot [1 - \Pr\{Z_j = 1\}] \\ &\quad = \bar{p} + (p + q - 1) \cdot \Pr\{Z_j = 1\}. \end{aligned} \tag{4}$$

Fig. 2. DTMC for: $p = q = 1$

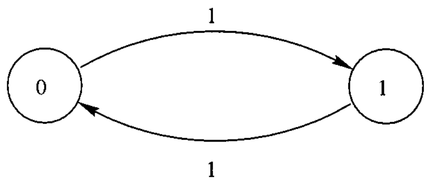

Fig. 3. DTMC for: $p = q = 0$

If $p + q = 1$, then the Markov chain describes a sequence of s-independent Bernoulli trials, and (4) reduces to:

$$\Pr\{Z_{j+1} = 1\} = \bar{p} = q,$$

which means that the failure probability does not depend on the outcome of the previous run. Thus each subsequent run succeeds or fails independently with probabilities p and $q = \bar{p}$ respectively. In this case, S_n defined by (1) is a sum of n mutually s-independent Bernoulli r.v. and has binomial pmf.

If $p + q \neq 1$ then the DTMC describes the sequence of s-dependent Bernoulli trials and enables us to accommodate possible s-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 (4) and the assumptions of binomial distribution do not hold. The pmf of S_n, given by the sum of n s-dependent r.v., can be derived using the observation that each visit to a given state of DTMC is a possibly delayed recurrent event [12].

When $p + q > 1$, runs are positively correlated: if a software failure occurs in run j, then there is an increased chance that another failure occurs in the next run. Thus, in this case, failures occur in clusters. The boundary case arises when equality in (3) holds: $p + q = 2$ ($p = q = 1$). This means that if the sequence of software runs starts with failure, then all successive runs fail; if it starts with success, then all successive runs succeed. That is, the Markov chain remains forever in its initial state as shown in Fig. 2.

Next consider the case when successive software runs are negatively correlated $p + q < 1$. In this case if a software failure occurs in run j, then there is an increased chance that a success occurs in run $(j+1)$, i.e., there is a lack of clustering. In the boundary case, when the equality in (3) holds ($p + q = 0$, i.e., $p = q = 0$), then the Markov chain alternates deterministically between the 2 states, as in Fig. 3.

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:

- $|p + q - 1| < 1$;
- DTMC in Fig. 1 is irreducible and aperiodic [34].

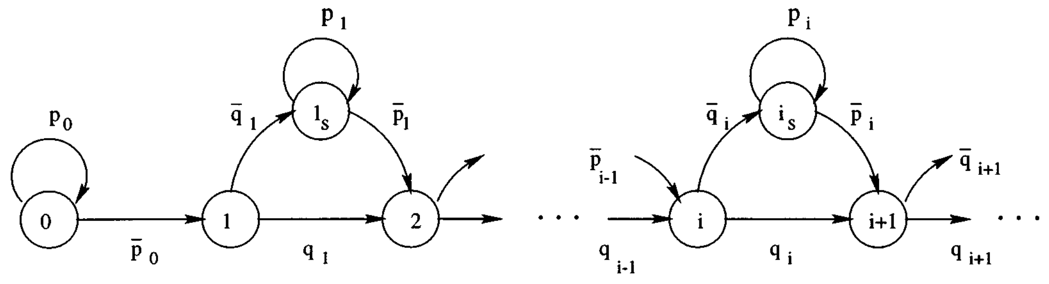

Fig. 4. Nonhomogeneous DTMC for SRGM

B. SRM 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. $F_{k,l}(t)$ is a Cdf of the time spent in a transition from state k to state l of the DTMC in Fig. 1. It seems realistic to assume that the runs execution times are not identically distributed for successful and failed runs. Thus, the $F_{k,l}(t)$ depend only of the type of point at the end of the interval:

$$F_{0,0}(t) = F_{1,0}(t) = F_{\text{ex}_S}(t),$$
$$F_{0,1}(t) = F_{1,1}(t) = F_{\text{ex}_F}(t).$$

With the addition of the $F_{k,l}(t)$ to the transitions of DTMC, we obtain an MRP as the SRM in continuous time. The $\{N(t), t \geq 0\}$ is a superposition of 2 s-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 $(0, t]$.

C. SRGM Based On MRP

In addition to assumptions #1–#3, this section uses assumptions #A1–#A4.

Assumptions:

A1) Whenever a failure occurs, faults are detected and removed instantaneously.

A2) At each such event, the probabilities of success and failure change.

A3) Software execution times are s-identically distributed for successful and failed runs.

A4) Run's execution time distribution does not change during the testing phase.

Assumptions A1, A3, A4 can easily be relaxed or removed as shown in Section III-D.

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, then an attempt is made to fix the underlying fault which causes the conditional probabilities of success and failure on the next run to change. Thus

$$P_i = \begin{bmatrix} p_i & \bar{p}_i \\ \bar{q}_i & q_i \end{bmatrix} \tag{5}$$

defines the p_i, q_i for the testing runs that follow the occurrence of failure i, up to the occurrence of the next failure, $(i+1)$. The SRGM in discrete time can be described with a sequence of s-dependent Bernoulli trials with step-dependent probabilities. The underlying stochastic process is a nonhomogeneous DTMC.

The sequence $S_n = Z_1 + \cdots + Z_n$ provides an alternate description of the SRGM considered here. Thus $\{S_n\}$ defines the DTMC presented in Fig. 4. Both states, i and i_s, represent that the failure state has been occupied i times. State i represents the first trial for which $S_n = i$. State i_s represents all subsequent trials for which $S_n = i$, i.e., all subsequent successful runs before the occurrence of next failure $(i+1)$. Without loss of generality let the first run be successful which means that 0 is the initial state.

The SRGM in continuous time is obtained by assigning runs execution-time distributions to transitions of the DTMC in Fig. 4. For simplicity, we have chosen the same execution time distribution regardless of the outcome:

$$F_{\text{ex}}(t) = F_{\text{ex}_S}(t) = F_{\text{ex}_F}(t), \quad \text{as in assumption #A3.}$$

Thus, T_{ex} of each software run has Cdf

$$F_{\text{ex}}(t) = \Pr\{T_{\text{ex}} \leq t\}.$$

Considering the situation when software execution times are not identically distributed for successful and failed runs,

$$F_{\text{ex}_S}(t) \neq F_{\text{ex}_F}(t)$$

is straightforward, and is discussed in Section III-D-1.

In 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 (failures) are of interest. Therefore, it is only necessary to consider the distribution of an interval between successive failures and to use the standard results of renewal theory. 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 Sf associated with failure $(i+1)$.

Derive the distribution of the discrete r.v. X_{i+1}, the number of runs between failures i and $(i+1)$. From Fig. 4, the r.v. $X_{i+1} (i \geq 1)$ has the pmf:

$$\Pr\{X_{i+1} = k\} = \begin{cases} q_i & \text{if } k = 1 \\ \bar{q}_i \cdot p_i^{k-2} \cdot \bar{p}_i & \text{if } k \geq 2. \end{cases} \tag{6}$$

Derive the distribution of the continuous r.v. T_{i+1}, which is the interval distribution of the point process $N_F(t)$ that records the number of failures in $(0, t]$.

It follows from (6) that the conditional Cdf of the time to failure $(i+1)$, given that the system has had i failures, for $i \geq 1$ is:

$$F_{i+1}(t) = q_i \cdot F_{\text{ex}}(t) + \sum_{k=2}^{\infty} \overline{q}_i \cdot p_i^{k-2} \cdot \overline{p}_i \cdot F_{\text{ex}}^{k*}(t) \quad (7)$$

$F_{\text{ex}}^{k*} \equiv k$-fold convolution of F_{ex}.

The LST of $F_{\text{ex}}(t)$ is the $\tilde{F}_{\text{ex}}(s)$ defined for $s \geq 0$ by

$$\tilde{F}_{\text{ex}}(s) = \int_0^{\infty} \exp(-s \cdot t)\, dF_{\text{ex}}(t) = E[\exp(-s \cdot t)].$$

The LST of $F_{i+1}(t)$ becomes

$$\begin{aligned}\tilde{F}_{i+1}(s) &= q_i \cdot \tilde{F}_{\text{ex}}(s) + \sum_{k=2}^{\infty} \overline{q}_i \cdot p_i^{k-2} \cdot \overline{p}_i \cdot \tilde{F}_{\text{ex}}^{k}(s) \\ &= \frac{q_i \cdot \tilde{F}_{\text{ex}}(s) + (1 - p_i - q_i) \cdot \tilde{F}_{\text{ex}}^2(s)}{1 - p_i \cdot \tilde{F}_{\text{ex}}(s)}. \quad (8)\end{aligned}$$

The inversion of (8) is, in principle, straightforward, and reasonably simple closed-form results can be obtained when $F_{\text{ex}}(t)$ has a rational LST (such as phase-type distributions [4]).

Some general properties of the inter-failure time are developed without making assumptions about the form of the run's execution-time $F_{\text{ex}}(t)$. Due to the well known property of LST, the moments can be derived easily by a simple differentiation of (8). Thus, the mean time to failure is:

$$\begin{aligned}E[T_{i+1}] &= - \left. \frac{d\tilde{F}_{i+1}(s)}{ds} \right|_{s=0} \\ &= - \frac{\overline{p}_i + \overline{q}_i}{\overline{p}_i} \cdot \left. \frac{d\tilde{F}_{\text{ex}}(s)}{ds} \right|_{s=0} \\ &= \frac{\overline{p}_i + \overline{q}_i}{\overline{p}_i} \cdot E[T_{\text{ex}}] \quad (9)\end{aligned}$$

$E[T_{\text{ex}}] \equiv$ the run's mean execution-time.

On average, there is growth in reliability:

$$E[T_{i+1}] > E[T_i]\text{—iff} \quad \frac{\overline{q}_{i+1}}{\overline{p}_{i+1}} > \frac{\overline{q}_i}{\overline{p}_i}.$$

If $p_i + q_i = 1$, then successive runs are s-independent and (9) becomes:

$$E[T_{i+1}^{i \cdot n}] = \frac{E[T_{\text{ex}}]}{\overline{p}_i}.$$

Thus, rewrite (9) as:

$$E[T_{i+1}] = (\overline{p}_i + \overline{q}_i) \cdot E[T_{i+1}^{i \cdot n}] = (1 - \pi_i) \cdot E[T_{i+1}^{i \cdot n}]. \quad (10)$$

Next, derive

$$E[T_{i+1}^2] = \left. \frac{d^2 \tilde{F}_{i+1}(s)}{ds^2} \right|_{s=0};$$

then substitute it together with (9) in:

$$\text{Var}[T_{i+1}] = \sigma_{T_{i+1}}^2 = E[T_{i+1}^2] - (E[T_{i+1}])^2.$$

Thus,

$$\begin{aligned}\text{Var}[T_{i+1}] = &\frac{\overline{p}_i + \overline{q}_i}{\overline{p}_i} \cdot \text{Var}[T_{\text{ex}}] \\ &+ \frac{(p_i + q_i) \cdot \overline{q}_i}{\overline{p}_i^2} \cdot (E[T_{\text{ex}}])^2. \quad (11)\end{aligned}$$

Besides the mean and variance, an important feature of distributions is the coefficient of variation:

$$C_{T_{i+1}} \equiv \frac{\sigma_{T_{i+1}}}{E[T_{i+1}]}. \quad (12)$$

$C_{T_{i+1}}$ is a relative measure of the spread of the distribution; the idea is that a deviation of amount $\sigma_{T_{i+1}}$ is less important, the larger $E[T_{i+1}]$ is. Substitute (9) and (11) into (12):

$$\begin{aligned}C_{T_{i+1}}^2 &= \frac{1}{\overline{p}_i + \overline{q}_i} \cdot C_{T_{i+1}^{i \cdot n}}^2 + \frac{(p_i - q_i) \cdot (p_i + q_i - 1)}{(\overline{p}_i + \overline{q}_i)^2} \\ &= \frac{1}{1 - \pi_i} \cdot C_{T_{i+1}^{i \cdot n}}^2 + \frac{(p_i - q_i) \cdot \pi_i}{(1 - \pi_i)^2}. \quad (13)\end{aligned}$$

$C_{T_{i+1}^{in}} \equiv$ is the coefficient of variation when the successive runs are s-independent.

A very plausible assumption for any software is: $p_i \gg q_i$. A program for which this assumption couldn't be made, would be enormously unreliable, and it is unlikely that its use would be ever attempted. It follows that the sign of the second term in (13) depends only on π_i.

The derived equations for the mean time to failure (10) and coefficient of variation (13) can show the effects of failure correlation on software reliability. If the successive software runs are s-dependent $(\pi_i \neq 0)$, then 2 cases are considered:

1) If $0 < \pi_i < 1$ $(p_i + q_i > 1)$ then successive runs are positively correlated: the failures occur in clusters. It follows from (10) that the mean time between failures is shorter than it would be if the runs were s-independent:

$$E[T_{i+1}] < E[T_{i+1}^{i \cdot n}].$$

The second term in (13) is positive, hence

$$C_{T_{i+1}}^2 > \frac{1}{1 - \pi_i} \cdot C_{T_{i+1}^{i \cdot n}}^2 > C_{T_{i+1}^{i \cdot n}}^2.$$

These results imply that in the presence of failure clustering, the inter-failure time has smaller mean and greater coefficient of variation compared to the s-independent case. That is, SRGM that assume s-independence among failures result in overly optimistic predictions.

2) If $-1 < \pi_i < 0$ $(p_i + q_i < 1)$ then successive runs are negatively correlated:

$$E[T_{i+1}] > E[T_{i+1}^{i \cdot n}].$$

The second term in (13) is negative; hence

$$C_{T_{i+1}}^2 < \frac{1}{1 - \pi_i} \cdot C_{T_{i+1}^{i \cdot n}}^2 < C_{T_{i+1}^{i \cdot n}}^2.$$

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

D. Model Generalizations

The model here can be generalized in many ways. Some of these generalizations are presented, together with brief comments on the nature of new ideas involved.

1) Replace Assumption #A3 With Assumption #4: Consider nonidentically distributed execution times for successful and failed runs $F_{\mathrm{ex}_S}(t) \neq F_{\mathrm{ex}_F}(t)$. Assign

- $F_{\mathrm{ex}_S}(t)$ to transition to state 0 and to all transitions to states i_S (successful runs),
- $F_{\mathrm{ex}_F}(t)$ to all transitions to state i, $i \geq 1$ (failed runs) of the DTMC in Fig. 4.

Make the appropriate changes in (7):

$$F_{i+1}(t) = q_i \cdot F_{\mathrm{ex}_F}(t) + \sum_{k=2}^{\infty} \overline{q}_i \cdot p_i^{k-2} \cdot F_{\mathrm{ex}_S}^{(k-1)*}(t) \cdot \overline{p}_i \cdot F_{\mathrm{ex}_F}(t) \qquad (14)$$

This leads to LST transform:

$$\tilde{F}_{i+1}(s) = \frac{q_i \cdot \tilde{F}_{\mathrm{ex}_F}(s) + (1 - p_i - q_i) \cdot \tilde{F}_{\mathrm{ex}_F}(s) \cdot \tilde{F}_{\mathrm{ex}_S}(s)}{1 - p_i \cdot \tilde{F}_{\mathrm{ex}_S}(s)}. \qquad (15)$$

2) Change Assumption A4: We have assumed that $F_{\mathrm{ex}}(t)$ does not change during the whole testing phase. This assumption can be violated for reasons such as important changes in the code due to the fault-fixing attempts. Thus, consider the situation when the parameters of $F_{\mathrm{ex}}(t)$ change after each failure occurrence (after each visit to the failure state). Then we need only to replace $F_{\mathrm{ex}}(t)$ with $F_{\mathrm{ex}_i}(t)$ in (7).

3) Remove Assumption A1: A DTMC which describes SRM in discrete time can be defined so that it considers noninstantaneous fault removal, periods of time when the software is idle, or more than one failure type. For example, DTMC in Fig. 5 can be used to describe various testing scenarios, such as:

1) $u = v = 1$: the model reduces to the one considered previously, i.e., instantaneous fault removal which occurs immediately after the failure is observed.
2) $u = 1$ and $v = 0$: noninstantaneous fault removal which occurs immediately after the failure is observed.
3) $u = 1$ and $v \neq 0$: delayed and noninstantaneous fault removal following the failure occurrence.
4) $u \neq 1$: the program can be changed, due to the fault removal or other modifications, following successful runs as well.

As for SRGM in Section III-C, the transition probabilities change at each visit to fault removal state.

With a few exceptions, such as [8], [9], the existing SRGM assume that faults that are source of failures are detected and corrected instantly following the failure occurrence, i.e., ignore repair times. The model in Fig. 5 allows assumption A1 to be removed in order to encompass the case when the fault removal process is noninstantaneous and does not necessarily follow immediately after the failure is observed. Unfortunately, unlike the run's execution time distribution which can be easily incorporated into the model based on measurements, currently there is a lack of information and little agreement about repair time distributions.

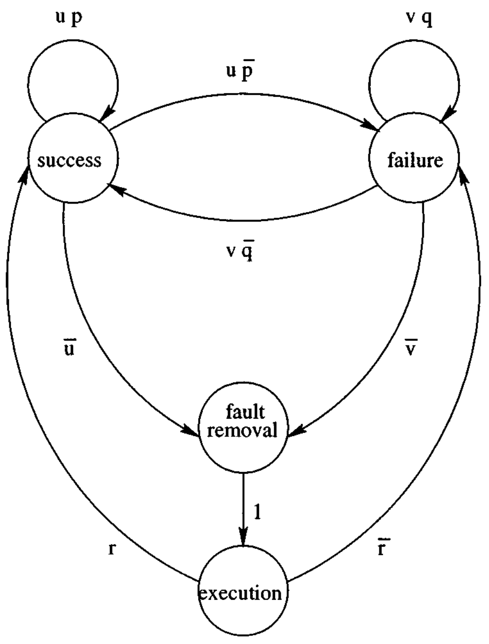

Fig. 5. DTMC for Non-Instantaneous and Delayed Fault Removal

IV. Applicability to Validation and Operational Phases

The MRP can be used to estimate reliability of the software, based on the results of testing performed in its validation phase. For 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, then the fault must be identified and removed. After the testing (debugging) phase the software enters a validation phase 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 a specified:

- number of test cases, or
- period of working,

that must be executed failure-free. While of particular interest for 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 s-independent Bernoulli trials, each run fails with probability $\theta = q$, i.e., succeeds with probability $1-\theta = p$, s-independently of the previous runs. Thus, the probability that n s-independent test runs are conducted without failure is:

$$(1-\theta)^n = \overline{q}^n = p^n. \qquad (16)$$

The largest value of θ such that

$$(1-\theta)^n \geq \alpha \qquad (17)$$

defines the $1-\alpha$ upper CB on θ [11], [34]. Solving (17) for θ gives $1-\alpha$ s-confidence that the failure probability per execution of a software that reveals no failure in n s-independent runs is below

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

For related recent work on a Bayes estimation approach see [19], [21].

Now consider a sequence of possibly s-dependent software runs. During the validation phase, the software is not changing: p and q do not vary. That is, the sequence of runs can be described by the homogeneous DTMC with transition probability matrix (2). Assume that the DTMC is stationary, i.e., each run has the same failure-probability:

$$\Pr\{Z_{j+1} = 1\} = \Pr\{Z_j = 1\} = \theta.$$

Then, from (4) it follows that

$$\theta = \frac{\overline{p}}{\overline{p} + \overline{q}}. \tag{19}$$

The probability that n successive runs succeed is:

$$\begin{aligned} &\Pr\{Z_n = \cdots = Z_2 = Z_1 = 0 | Z_0 = 1\} \\ &\quad = \Pr\{Z_n = 0 | Z_{n-1} = 0\} \cdots \Pr\{Z_2 = 0 | Z_1 = 0\} \\ &\quad\ \cdot \Pr\{Z_1 = 0 | Z_0 = 1\} \\ &\quad = p^{n-1} \cdot \overline{q}. \end{aligned} \tag{20}$$

Use (19) to rewrite (20) in terms of θ and π. It follows that in the case of s-dependent Bernoulli trials $(1-\alpha)$ upper CB on failure probability per execution θ becomes:

$$\theta^* = \frac{1 - \alpha^{1/(n-1)}}{1 - \pi}. \tag{21}$$

If failures occur in clusters $(0 < \pi < 1)$ then the CB on failure probability is approximately $1/(1-\pi)$ higher than the CB obtained under the s-independence assumption (18). It follows that the result obtained under the assumption of s-independence is overly optimistic.

Consider the applicability of the MRP approach in the operational phase. During the operational phase, assume that no changes are made to the software. Then the sequence of runs can be described by the homogeneous DTMC. The well developed theory of MRP [4], [26] and its limiting results can be used to derive several measures other than the distribution of the time between failures, such as the:

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

The MRP approach can be used for modeling fault-tolerant software systems. Per-run failure probability and run's execution-time distribution for a particular fault-tolerant technique can be derived using a variety of existing models (see [10], [20], [30] and references therein). Thus, in addition to the interversion failure correlation on a single run considered in related work, our approach can account for the correlation among successive failures.

V. Some Special Cases and Relation to Existing Models

The MRP approach in this paper can be related to existing SRM. The SRM in discrete time is comparable with input-domain based models which are static and don't consider the time; while the SRM in continuous time is comparable with time-domain SRGM. We next examine these relations.

The SRM in discrete time (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 [24] defines:

$$\lambda_i (0 \le \lambda_i \le 1)$$

as a fault-size under operational inputs after change i to the program, i.e., given that i faults have been detected and corrected. Then the probability that there are $k \ge 0$ successful runs before the occurrence of the next failure, $(i+1)$, is

$$P_i\{k | \lambda_i\} = (1 - \lambda_i)^k \cdot \lambda_i. \tag{22}$$

Eq. (22) is a special case of (6) under the assumption that successive runs are s-independent. In [24], λ_i is treated as a r.v. with a Cdf $G(\lambda_i)$; (22) becomes:

$$P_i(k) = \int (1 - \lambda_i)^k \cdot \lambda_i dG(\lambda_i) = E[(1 - \lambda_i)^k \cdot \lambda_i].$$

The SRM in discrete time is also related to the Compound-Poisson SRM in [27]. This model is based on the work in [28] which approximates the sum of s-dependent integer-valued r.v. S_n, given by (1) with a compound Poisson process when these r.v. are rarely nonzero and given they are nonzero their conditional distributions are nearly identical.

Consider the SRM in continuous time. If $\tilde{F}_{\text{ex}}(s)$ 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 $f_{\text{ex}}(t) = \mu \cdot \exp(-\mu \cdot t)$, since it relates the MRP approach to the existing time-domain based SRGM and some of their basic assumptions, such as the s-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 s-independent $(p_i + q_i = 1)$,
- software execution time is exponentially distributed with rate μ.

Inverting (8) leads to the pdf of the inter-failure times:

$$f_{i+1}(t) = \overline{p}_i \cdot \mu \cdot \exp[-\overline{p}_i \cdot \mu \cdot t]; \tag{23}$$

i.e., the conditional reliability is

$$R_{i+1}(t) = 1 - F_{i+1}(t) = \exp[-\overline{p}_i \cdot \mu \cdot t]. \tag{24}$$

It follows that the inter-failure time is exponentially distributed with rate $\overline{p}_i \cdot \mu$ if the software testing is considered as a sequence of s-independent runs with an exponential distribution of the execution times.

The alternative interpretation of this special case is in [15]. Under the assumption that inputs arrive at the 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:

$$1 - F(t) = \sum_{j=0}^{\infty} \frac{\exp(-\mu \cdot t) \cdot (\mu \cdot t)^j}{j!} \cdot \left(\frac{M - M^*}{M} \right)^j \quad (25)$$

M is the size of the input data space (number of data items which can be used as input to the software), M^* is the total number of these input data which can cause software failure. The first term inside the summation is the probability that j inputs are received in time t; the second term is probability that none of j inputs lead to a failure of software. Eq. (25), when simplified, leads to:

$$1 - F(t) = \exp\left[-\frac{M^*}{M} \cdot \mu \cdot t \right]. \quad (26)$$

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

The conditional reliability (survivor function of the time between failures $(i-1)$ and i, T_i) becomes

$$1 - F_i(t) = \Pr\{T_i > t\} = \exp(-\lambda_i \cdot t). \quad (27)$$

Even though the motivations are different and the parameters have different interpretations, mathematically the model in [15] is a special case of the SRM based on MRP under the assumptions that successive software runs are s-independent and the software execution times are exponentially distributed with rate μ. Ref [15]– shows that the JelinskiMoranda model [14] can be obtained by introducing

$$\lambda_i = \frac{\mu}{M} \cdot (N - i + 1) = \Lambda(N - i + 1)$$

and treating Λ and N– as unknown parameters. Then by adopting a Bayes point of view, two other models can be derived from the JelinskiMoranda model [15]:

1) Goel–Okumoto model [7] is obtained by assuming Λ has a known value, and assigning Poisson prior distribution for N;
2) Littlewood–Verrall model [17] 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 s-independence. For example, the Moranda model [22] also assumes s-independent exponentially distributed times between failures with a failure rate that decreases in geometric progression on the occurrence of each individual failure:

$$\lambda_i = \lambda k^{i-1}, \qquad k < 1.$$

Due to space limitations and the vast number of SRGM, the analysis of the relation to the existing models is not pursued further.

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

$$f_{i+1}(t) = \frac{\overline{q}_i}{p_i} \cdot \overline{p}_i \cdot \mu \cdot \exp[-\overline{p}_i \cdot \mu \cdot t] + \frac{(p_i + q_i - 1)}{p_i} \cdot \mu \cdot \exp[-\mu \cdot t]. \quad (28)$$

This distribution is a mixture (compound) distribution [34] with pdf of form:

$$g(t) = \alpha_1 \cdot g_1(t) + \alpha_2 \cdot g_2(t) \qquad \alpha_1,\, \alpha_2 > 0, \quad \alpha_1 + \alpha_2 = 1.$$

When $p_i + q_i > 1$ the inter-failure distribution (28) is hyperexponential: a mixture of two exponential distributions with rates

$$\overline{p}_i \cdot \mu, \quad \text{and} \quad \mu.$$

The mixing probabilities are

$$\alpha_1 = \overline{q}_i / p_i,$$
$$\alpha_2 = (p_i + q_i - 1)/p_i,$$

respectively. The coefficient of variation for the hyperexponential distribution is greater than 1. Due to the presence of failure clustering, the inter-failure time has smaller mean and greater variance, compared to the s-independent case, even under the assumption of exponentially distributed duration of testing runs.

When $p_i + q_i < 1$ then (28) becomes a mixture of an exponential distribution with rate $\overline{p}_i \cdot \mu$ and hypoexponential distribution with rates $\overline{p}_i \cdot \mu$ and μ. The mixing proportions are

$$\alpha_1 = q_i / \overline{p}_i,$$
$$\alpha_2 = (1 - p_i - q_i)/\overline{p}_i,$$

respectively. The coefficient of variation is less than 1. Thus, the inter-failure time has greater mean and smaller variance compared to the s-independent case.

These results demonstrate the effects of failure correlation on the software reliability measures. Some of the common assumptions made by SRM are inadequate and result in optimistic estimations when failures are clustered.

VI. Discussion and Future Work

The ultimate goal when developing SRGM is 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 aimed mainly at showing that the classical software reliability theory can be extended to consider a sequence of possibly s-dependent software runs, *viz,* failure correlation. However, there are many research issues that we want to address in the near future in order for the model to be fully specified and applied in real software development projects for estimating and predicting.

Consider in some detail the concept of software runs. The operation of software can be broken into series of runs [23]. Each software run performs mapping between a set of input variables and a set of output variables, and consumes execution time. The input variable for a software run is any data item that exists external to the software run and is used by a software run. There does not have to be a physical input process. The input variable can simply be located in memory, waiting to be accessed. Even for software that operates continuously, it is possible and more convenient to divide the operation into software runs by subdivision of time associated with some user-oriented tasks [23]. The information associated with software runs can generally be grouped into 2 categories:

- *Timing:* This includes specific time associated with each software 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 software 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 software runs, as well as for successful software runs are routinely captured in many projects for test tracking, and testing process management purposes [31], [32]. These data, with possibly minor modifications, provide common source for SRGM, input-domain analysis, and integrated analysis in this paper. The MRP approach considers all the knowledge that we have about system: the outcomes and execution times of all testing runs, either successful or failed.

The existing time-domain SRGM disregard the successful runs between two failures and ignore the information conveyed by them. Successful runs (nonfailure stops of software operation) are considered in [2] for parameter estimation of some of the existing SRGM. It is pointed out that 'disregarding the non-failure stops' violates the Maximum Information Principle that suggests exploiting available software reliability information as much as possible.

For the MRP approach, the T_{i+1} is a r.v. whose distribution depends on the distribution of the run's execution time $F_{\text{ex}}(t)$, and on the conditional probabilities p_i, 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 Cdf of the run's execution time $F_{\text{ex}}(t)$ can be determined from measured data.

Consider possible models for the parameter set $\{p_1, q_1, p_2, q_2, \cdots\}$.

It is mathematically possible to permit an infinite number of failures; in this case the parameter set is infinite. By setting

$$q_i = 0, \quad p_i = 1, \qquad \text{for } i \geq n+1,$$

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. There are two possible approaches to modeling the parameter set $\{p_i, q_i\}$:

- Relate them to the number of test runs or to the number of faults in a systematic way by assuming various deterministic functional relationships. The free parameters are seen simply as unknown constants to be estimated by the sample data.
- Treat the parameters as r.v. themselves, to consider the uncertainty about the effect of fault removal [17], [18]. Thus, even if the fault removal is successful, there is uncertainty about the size of the fault removed, and thus uncertainty about the magnitude of reliability improvement. This approach results in a doubly stochastic model: one model for set of parameters; the other for the times between failures, conditional on the parameters.

The brief review of SRGM discussed in previous section is presented to demonstrate the differences that arise from the additional assumptions about the way the modeling parameters change as a result of the fault removal attempts. The models, Jelinski–Moranda [14], Moranda [22], and Littlewood–Verrall [17], are examples of TBF models and all have the common assumption that the inter-failure times are s-independent exponentially distributed r.v.

$$f_{i+1}(t) = \lambda_i \cdot \exp(-\lambda_i \cdot t).$$

They differ in assumptions about the way λ_i changes as a result of fault removal attempts.

The Jelinski–Moranda model assumes that the initial number of faults N is an unknown fixed constant, faults are instantly and perfectly corrected without introducing any new faults, and all faults that still remain are equally likely to occur. As a consequence, failure intensity is proportional to the residual number of faults,

$$\lambda_i = \Lambda(N - i + 1):$$

which means that all faults have the same size.

The Moranda model assumes that the initial number of faults is infinite and that the faults detected and removed early in the software testing decrease failure intensity more rapidly from those detected later on. Thus the failure intensity decreases in a geometric progression on the occurrence of each individual failure,

$$\lambda_i = \lambda \cdot k^{i-1}, \qquad (k < 1).$$

The Littlewood–Verrall model is a doubly stochastic model which treats the λ_i as a r.v. with a gamma distribution. This reflects the likelihood, but not a guarantee, that a fault removal improves reliability and, if an improvement occurs, it is of uncertain magnitude.

The Goel–Okumoto model [7] treats N, initial number of faults in software system, as a Poisson r.v. which results in NHPP with failure intensity $\lambda(t)$. However, failure occurrence rate per fault for this model is constant, that is, all faults have same size. In contrast to the constant fault detection rate of the Goel–Okumoto model, many commonly used FC models are obtained by choosing different $\lambda(t)$ which results in different NHPP that can capture increasing/decreasing failure occurrence rate per fault. The SRGM based on NHPP differ from TBF models in one more aspect: the inter-failure times are not s-independent and the nonstationarity of the process complicates their distributions. Nevertheless, if NHPP has failure intensity $\lambda(t)$ then, given that there are n failures in interval $(0, t]$, these failures are i.i.d. with pdf [3]

$$\frac{\lambda(t)}{\displaystyle\int_0^t \lambda(u)\, du}.$$

To summarize, while SRGM can be related to each other and derived from the same mathematical framework, they may differ appreciably in their assumptions about the failure mechanisms, fault removal process, and overall growth in reliability during the testing phase. The software reliability modeling framework in this paper provides the basis for a more flexible, consistent approach to the mathematical formulation of software reliability, and contributes toward more realistic modeling since it integrates the phenomena of failure correlation. 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. Some basic issues that should be taken into account when making additional assumptions about the

- fault-removal process,
- way that modeling parameters change as a result of fault removal attempts,

Ref. [1], [35] are briefly outlined here:

- *Size of faults:* In general, different software faults do not affect the failure probability equally. 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. An attempt to fix one fault can introduce new faults in the 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:* The failure history of a program depends on the testing strategy used; so SRGM must consider the testing process used. In practice it is important to deal with nonhomogeneous testing, i.e., the model should include the possibility of describing variations of the testing strategy with time.

References

[1] S. Bittanti, P. Bolzern, and R. Scattolini, “An introduction to software reliability modeling,” , vol. 29, pp. 43–66.

[2] K. Cai, “Censored software-reliability models,” *IEEE Trans. Reliability*, vol. 46, no. 1, pp. 69–75, Mar. 1997.

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

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

[5] L. H. Crow and N. D. Singpurwalla, “An empirically developed Fourier series model for describing software failures,” *IEEE Trans. Reliability*, vol. R-33, pp. 176–183, June 1984.

[6] W. Farr, “Software reliability modeling survey,” , vol. 13, pp. 71–117.

[7] A. L. Goel and K. Okumoto, “Time dependent error-detection rate model for software reliability and other performance measures,” *IEEE Trans. Reliability*, vol. R-28, pp. 206–211, 1979.

[8] S. Gokhale, P. N. Marinos, K. S. Trivedi, and M. R. Lyu, “Effect of repair policies on software reliability,” *Proc. Computer Assurance (COMPASS’1997)*, pp. 105–116, 1997.

[9] S. Gokhale, M. Lyu, and K. Trivedi, “Software reliability analysis incorporating fault detection and debugging activities,” in *Proc. 9th IEEE Int’l Symp. Software Reliability Engineering*, 1998, pp. 202–211.

[10] K. Goševa-Popstojanova and A. Grnarov, “Hierarchical decomposition for estimating reliability of fault-tolerant software in mission-critical systems,” in *Proc. IASTED Int’l. Conf. Software Engineering*, 1997, pp. 141–146.

[11] D. Hamlet, “Are we testing for true reliability?,” *IEEE Software*, pp. 21–27, July 1992.

[12] J. J. Hunter, “Mathematical techniques of applied probability,” in *Discrete Time Models: Techniques and Applications*: Academic Press, 1983, vol. 2.

[13] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*: McGraw-Hill, 1996.

[14] Z. Jelinski and P. B. Moranda, “Software reliability research,” in *Statistical Computer Performance Evaluation*, W. Freiberger, Ed: Academic Press, 1972, pp. 485–502.

[15] N. Langberg and N. D. Singpurwalla, “A unification of some software reliability models,” *SIAM J. Sci. Stat. Comput.*, vol. 6, pp. 781–790, July 1985.

[16] J. Laprie and K. Kanoun, “Software reliability and system reliability,” , vol. 13, pp. 27–69.

[17] B. Littlewood and J. L. Verrall, “A Bayesian reliability growth model for computer software,” in *Proc. IEEE Symp. Computer Software Reliability*, 1973, pp. 70–77.

[18] B. Littlewood, “Modeling growth in software reliability,” in *Software Reliability Handbook*, P. Rook, Ed: Elsevier Applied Science, 1990, pp. 137–153.

[19] B. Littlewood and D. Wright, “Some conservative stopping rules for the operational testing of safety-critical software,” *IEEE Trans. Software Engineering*, vol. 23, pp. 673–683, Nov. 1997.

[20] D. F. McAllister and M. A. Vouk, “Fault-tolerant software reliability engineering,” , vol. 13, pp. 567–614.

[21] K. W. Miller, L. J. Morell, and R. E. Noonan *et al.*, “Estimating the probability of failure when testing reveals no failures,” *IEEE Trans. Software Engineering*, vol. 18, pp. 33–43, Jan. 1992.

[22] P. B. Moranda, “Prediction of software reliability during debugging,” in *Proc. Annual Reliability and Maintainability Symp*, 1975, pp. 327–332.

[23] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*: McGraw-Hill, 1987.

[24] C. V. Ramamoorthy and F. B. Bastani, “Modeling of the software reliability growth process,” *Proc. COMPSAC*, pp. 161–169, 1980.

[25] ——, “Software reliability—Status and perspectives,” *IEEE Trans. Software Engineering*, vol. SE-8, pp. 354–371, July 1982.

[26] S. M. Ross, *Applied Probability Models with Optimization Applications*: Holden-Day, 1970.

[27] M. Sahinoglu, “Compound-Poisson software reliability model,” *IEEE Trans. Software Engineering*, vol. SE-18, pp. 624–630, July 1992.

[28] R. F. Serfozo, “Compound Poisson approximations for sums of random variables,” *Annals of Probability*, vol. 14, no. 4, pp. 1391–1398, 1986.

[29] S. Bittanti, Ed., *Software Reliability Modeling and Identification*: Springer-Verlag, 1988, vol. 341.

[30] A. T. Tai, J. F. Meyer, and A. Avižienis, “Performability enhancement of fault-tolerant software,” *IEEE Trans. Reliability*, vol. 42, pp. 227–237, June 1993.

[31] J. Tian, “Integrating time domain and input domain analyzes of software reliability using tree-based models,” *IEEE Trans. Software Engineering*, vol. 21, pp. 945–958, Dec. 1995.

[32] J. Tian and J. Palma, “Data partition based reliability modeling,” in *Proc. 7th IEEE Int’l Symp. Software Reliability Engineering*, 1996, pp. 354–363.

[33] L. A. Tomek, J. K. Muppala, and K. S. Trivedi, “Modeling correlation in software recovery blocks,” *IEEE Trans. Software Engineering*, vol. 19, pp. 1071–1086, Nov. 1993.

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

[35] M. Xie, *Software Reliability Modeling*: World Scientific Publishing Company, 1991.

Katerina Goševa-Popstojanova (M) received the B.S. (1980), M.S. (1985), and Ph.D. (1995) in Computer Science from the Faculty of Electrical Engineering, University “Sv. Kiril i Metodij Skopje,” Macedonia. Since 1997, she has been working as an Assistant Professor in the Department of Computer Science at Faculty of Electrical Engineering, University Sv. Kiril i Metodij. Currently she is a visiting scholar in the Department of Electrical and Computer Engineering at Duke University. Her research interests include software reliability, fault-tolerant computing, and dependability, performance, and performability modeling; she has published numerous articles on these topics. She is a Member of IEEE.

Kishor S. Trivedi (F) received the B.Tech. from the Indian Institute of Technology (Bombay), and M.S. and Ph.D. in Computer Science from the University of Illinois, Urbana-Champaign. He is the author of the well known text, *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* He has recently published two books: *Performance and Reliability Analysis of Computer Systems* and *Queuing Networks and Markov Chains.* His research interests are in reliability and performance assessment of computer and communication systems. He has published over 250 articles, and has lectured extensively on these topics. He has supervised 32 Ph.D. dissertations. He is an IEEE Fellow, and a Golden Core Member of the IEEE Computer Society. He holds the Hudson chair in the Department of Electrical and Computer Engineering at Duke University, and holds a joint appointment in the Department of Computer Science there. He is the Duke-Site Director of an NSF Industry-University Cooperative Research Center between NC State University and Duke University for carrying out applied research in computing and communications. He has been on the Duke faculty since 1975, and has served as a Principal Investigator on various AFOSR, ARO, Burroughs, Draper Lab, IBM, DEC, Alcatel, Telcordia, Motorola, NASA, NIH, ONR, NSWC, Boeing, Union Switch and Signals, NSF, and SPC funded projects, and as a consultant to industry and research laboratories. He was an Editor of IEEE TRANS. COMPUTERS during 1983–1987. He is a co-designer of HARP, SAVE, SHARPE, SPNP, and SREPT modeling packages; these packages have been widely circulated.