Estimating the probability of failure when software runs are dependent: An empirical study

Katerina Goseva-Popstojanova and Margaret Hamill
Lane Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV 26506-6109
E-mails: Katerina.Goseva@mail.wvu.edu, mhamill@mix.wvu.edu

***Abstract*—The assumption of independence among successive software runs, common to many software reliability models, often is a simplification of the actual behavior. This paper addresses the problem of estimating software reliability when the successive software runs are statistically correlated, that is, when an outcome of a run depends on one or more of its previous runs. First, we propose a generalization of our previous work using higher order Markov chain to model a sequence of dependent software runs. Then, we conduct an empirical study for exploring the phenomenon of dependent software runs using three software applications as case studies. Based on two statistical approaches, we show that the outcomes of software runs (i.e., success or failure) for two of the case studies are dependent on the outcome of one or more previous runs, in which case first or higher order Markov chain models are appropriate. Finally, we estimate the parameters of the appropriate models and discuss the effects of dependent software runs on the estimates of the software reliability.**

I. Introduction

Significant amount of research work has been done in developing software reliability models. However, exploring the validity of the assumptions and relaxing the restrictions have not received an adequate attention. Although some of the underlying assumptions may be unrealistic and do not apply universally, they still are used commonly due to the tractability of the resulting mathematical models. The lack of experimental data disproving these assumptions also contributes to their widespread use.

Among the basic assumptions made by various models, in this paper we address the validity of the assumption that successive software runs (i.e., executions), and consequently successive software failures, are mutually independent. (It should be noted that this is different from considering the independence of component failures within a single run addressed in [6] and references therein.)

As adaptive and real-time systems become more widely used, it is easy to understand why successive runs may not be independent. For example, in a real-time control system the sequence of input values to the software tend to change slowly, that is, successive inputs are very close to each other. For this reason, given a failure of a software for a particular input, there may be a greater likelihood of it failing for successive inputs. Similarly, in applications that operate on demand similar types of demands often tend to occur close together, which can also result in a series of related software runs.

Most software reliability models assume that software is tested by randomly selecting test cases throughout the input space. However, in practice this assumption may not be true. For example, test cases may be grouped by high level functionality or even by requirements. In such cases, a series of related (not independent) tests are run. Further, testing is usually conducted in such a way to increase the effectiveness, leading to detection of as many faults as possible. Thus, once a failure is observed a series of related tests are often run in order to help isolating the cause of failure.

Another reason that may lead to stochastic dependence of successive software runs is related to the extent to which the 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) [8].

To summarize, the assumption of independence of successive software runs may be violated in many practical situations. As a result, if a software failure occurs there is likely an increased chance that another failure will occur shortly there after. We say that software failures occur in clusters if failures have tendency to occur in groups, that is, if the times between successive failures are short for some periods of time and long for other periods of time.

In this paper we propose a generalization of our previous work on modeling a sequence of dependent software runs [5] to include higher order Markov chains. It should be noted that while the probability theory of Markov chains has been extensively developed, relatively little attention has been paid to statistical inference concerning Markov chain models, both in general and in software reliability estimation in particular. This is no surprise since the theory is elegant, whereas statistical inference is generally more difficult mathematically and also gives rather "messy" calculations in practice [1]. In this paper we present empirical analysis based on three software applications. We first use two statistical approaches to identify the order of the Markov chain. Then, we illustrate the effect of dependent software runs on the estimates of software reliability.

The remainder of this paper is organized as follows. The related work is presented in section II. The software

978-0-7695-3878-5/09 $26.00 © 2009 IEEE
DOI 10.1109/ISSRE.2009.24

reliability models capable of accounting for dependency among successive software runs are presented in section III, followed by the description of statistical methods used for testing the order of the Markov chain model given in section IV. The experimental setup and the case studies are described in section V. The empirical results are presented in section VI. Finally, the concluding remarks are given in section VII.

II. Related work

Software reliability models are used to both quantify the current reliability and predict future reliability. The assumption that successive software failures are independent is a standard assumption that applies to each software reliability growth model presented in [4]. According to [7] one of the reasons "Why conventional reliability theory fails" for software is that the runs are not always independent.

Failure correlation has not been explored extensively in the literature. In fact, to the best of our knowledge there are only a few published papers on the topic. The Fourier series model proposed in [2] can be used for analyzing clustered failure data, especially those with cyclic behavior. The Compound-Poisson software reliability model presented in [10] considers multiple failures that occur simultaneously in bunches within a specified time. Modeling the correlation between successive executions of the software fault-tolerance technique based on recovery blocks was presented in [11].

This paper builds on our previous work which proposed a software reliability modeling framework capable of incorporating the possible dependence among successive software runs [5]. Markov renewal process (MRP) formulation allows the model to be constructed in two stages. First, the outcomes (i.e., pass/fail) of successive software runs are used to construct a Discrete Time Markov Chain (DTMC) model. Then, considering the execution times of the software runs a model in continuous time is built. The result is a Semi Markov Process (SMP) which describes both failure and execution behavior. The model naturally introduces dependence among successive software runs, that is, the independence among software runs is a special case of the proposed model.

The work presented in [5] was generalized in [3]. Following the same approach based on Markov renewal processes, authors in [3] considered more than one type of failure, as well as the cases of restarting with repair and without repair. Neither our previous work [5] nor the generalization presented in [3] addressed the statistical inference of the proposed models and did not include empirical studies based on real software applications.

In this paper we focus on the generalization and empirical evaluation of the discrete time Markov model which represents the outcomes of a sequence of dependent software runs. Incorporating the software execution behavior through the distribution of software run's execution time, although fairly straightforward, is left as a future work.

III. Modeling dependent software runs

We start this section with introducing the notation and the basic definitions. Then, for completeness, we present a brief summary of our previous work which models a sequence of dependent software runs by a first order Markov chain in which the probabilities of the outcome of different runs depend only on the immediately preceding run. Then, we propose a generalization consisting of second and higher order Markov models.

Note that throughout the paper it is assumed that we have a steady-state behavior so that the probabilities of a software run passing or failing do not change with time. In statistical terms this means that the sequence of events is considered to be a stationary process.

A. Definitions and notation

The outcome of each software run can be treated as a binary random variable such that 0 represents a successful execution and 1 represents a failed execution. Let $P(i)$ denote the probability of outcome i ($i = 0, 1$). Let $P(i, j)$ denote a joint probability that in two successive runs the first outcome is i and the second outcome is j. Similar definition applies for $P(i, j, k)$ and other higher level joint probabilities.

The probability of observing an outcome j given that the previous outcome was i is called a transition probability and is denoted by $P(j|i)$. Note that textbooks and articles on stochastic processes usually denote the transition probabilities by p_{ij} [12]. In this paper, in addition to the standard p_{ij} notation, to avoid confusion we also use the conditional probability notation $P(j|i)$ in some places.

The successive events are independent if the transition probabilities do not depend on the preceding event, in which case $P(j|i) = P(j)$, for all i, j, which leads to $P(i, j) = P(i)P(j|i) = P(i)P(j)$, for all i, j.

If the independence model is not valid, a simple alternative for modeling a sequence of runs is a stationary first order Markov chain in which the probabilities of a different events depend on the immediately preceding event, but not on earlier events. Let $P(k|i, j)$ denote the conditional probability of observing an outcome k, given that the previous two outcomes were i and j. Thus, $P(k|i, j)$ actually represents another notation for the transition probability p_{ijk}. The sequence of events can be modeled with a first order Markov chain if $P(k|i, j) = P(k|j)$, for all i, j, k.

If a Markov chain of any order is assumed to be an appropriate model for a sequence of events, then the observed frequencies of different sequences may be used to estimate different probabilities and to determine the order of the Markov chain.

Let n_i denote the observed frequency of outcome i. Let n_{ij} denote the observed frequency of pairs of events in which outcome i is followed by outcome j. A similar definitions can be introduced for n_{ijk} and so on. It follows that the total number of software runs is $N_1 = \sum_i n_i$ and the total number of pairs is $N_2 = \sum_{i,j} n_{ij}$ and so on. We assume that software runs come in one sequence, in which case $N_1 = N_2 + 1 = N_3 + 2 = \ldots$ since the total number of runs is greater than the total number of pairs, which in turn is greater than the total number of triplets and so on.

In the statistical tests we need the marginal frequencies for which we use the notation in which a dot in the suffix implies summation over that suffix. For example, the number of pairs that start with outcome i is given by $n_{i.} = \sum_j n_{ij}$ while the number of triplets that finish with outcome k is $n_{..k} = \sum_{i,j} n_{ijk}$

B. First order Markov model

The simplest generalization of a sequence of independent software runs is to consider that the probability of a specific run passing or failing depends only on the outcome of the previous run. Let Z_m be a binary valued random variable that represents the outcome of the mth trial, such that 0 represents a successful execution and 1 represents a failed execution. The sequence of dependent Bernoulli trials $\{Z_m; m \geq 1\}$ defines a first order discrete time Markov chain shown in Figure 1, which has two states: 0 and 1 regarded as success and failure, respectively. Its transition probability matrix is given by

$$P = \begin{bmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{bmatrix}, \qquad 0 \leq p_{ij} \leq 1; \quad \sum_j p_{ij} = 1. \quad (1)$$

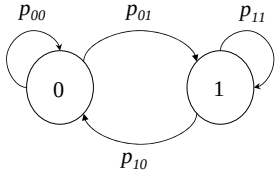

Figure 1. First order Markov model

The probability $p_{01} = P(1|0)$ ($p_{11} = P(1|1)$) is the conditional probability of failure on a software run given that the previous run has succeeded (failed). It can be shown that the unconditional probability of failure on the $(m+1)$st run is given by [5]:

$$P\{Z_{m+1} = 1\} = p_{01} + (p_{11} - p_{01})\, P\{Z_m = 1\}. \quad (2)$$

In the special case where $p_{01} = p_{11}$, Figure 1 actually represents a sequence of independent Bernoulli trials. Equation (2) reduces to $P\{Z_{m+1} = 1\} = p_{01} = p_{11}$, which simply means that the failure probability does not depend on the outcome of the previous run. In other words, each run has independent probabilities of succeeding or failing.

On the other hand, when $p_{01} \neq p_{11}$ the DTMC describes a sequence of dependent Bernoulli trials which accommodates dependence among successive executions. In this case the outcome of the software execution (success or failure) depends on the outcome of the previous run as in equation (2).

The relation between the conditional probabilities shows the presence or the lack of failure clustering. Specifically, if $p_{11} > p_{01}$, successive software runs are positively correlated, that is, if a failure occurs in mth run, there is an increased chance that a failure will occur in $(m+1)$st run. Obviously, this leads to failures clustering. Now consider the case when $p_{11} < p_{01}$, that is, successive software runs are negatively correlated. In this case, if a software failure occurs in mth run, there is an increased chance that a success will occur in the next run, and so there is a lack of clustering.

The boundary cases when $p_{00} = p_{11} = 1$ or $p_{00} = p_{11} = 0$ are excluded from the analysis since they are trivial, with no practical interest. This implies that the DTMC in Figure 1 is irreducible and aperiodic [12].

As described earlier p_{01} and p_{11} are conditional probabilities of failure, given that the previous run has succeeded and failed, respectively. Of interest here is to derive the unconditional probability of failure per run θ, which in this case is equal to the probability $P(1)$ that the Markov chain is in state 1. Since for a stationary chain $P\{Z_{m+1} = 1\} = P\{Z_m = 1\} = \theta$ from (2) it follows that

$$\theta = P(1) = \frac{p_{01}}{p_{01} + (1 - p_{11})} = \frac{p_{01}}{p_{01} + p_{10}}. \quad (3)$$

Of course, the same solution can be derived by solving the Markov chain defined by transition probability matrix (1).

C. Higher order Markov models

More generally the probabilities of the outcomes of different runs may depend on the r preceding events, that is, can be modeled with an rth order Markov chain. Any Markov chain of order r with c possible states can be regarded as a Markov chain of first order with c^r possible states specified by the set of r consecutive values of the original variable. Thus, in case of a sequence of software runs with two possible outcomes ($c = 2$) representing failure and pass, the second order Markov chain ($r = 2$) can be represented with first order chain with four states ((00), (01), (10) and (11)) and transition probability matrix given by:

$$P = \begin{bmatrix} p_{000} & p_{001} & 0 & 0 \\ 0 & 0 & p_{010} & p_{011} \\ p_{100} & p_{101} & 0 & 0 \\ 0 & 0 & p_{110} & p_{111} \end{bmatrix} \quad (4)$$

where $0 \leq p_{ijk} \leq 1$ and $\sum_k p_{ijk} = 1$.

In a second order Markov chain each event depends on the two immediately preceding events. Thus, $p_{ijk} = P(k|ij)$ in (4) denotes the conditional probability that the outcome k is

observed, given the previous outcomes were i and j, that is, the transition probability from state (ij) to state (jk) in the first order representation of the second order Markov chain. The state transition diagram which represents the second order Markov chain is presented in Figure 2.

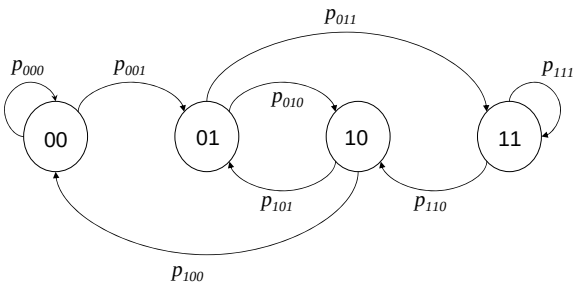

Figure 2. Second order Markov model

The unconditional probability of failure in any run in the case of the second order Markov chain is

$$\theta = P(01)+P(11) = \frac{p_{001}(p_{011}+p_{110})}{p_{001}(p_{011}+p_{110})+p_{110}(p_{001}+p_{100})} \quad (5)$$

It should be noted that in case when $P(k|ij) = P(k|j)$ (i.e., $p_{001} = p_{101}$ and $p_{011} = p_{111}$) the second order Markov chain reduces to a first order Markov chain in which case (5) reduces to (3).

In general, a sequence of software runs with rth order dependence can be modeled with r order Markov chain. Due to lack of space the corresponding equations are not presented in the paper, although they are used for estimation of the unconditional failure probability for one of the case studies in section VI.

IV. Tests for the Markov chain order

In practice the appropriate model for a given sequence of software runs is usually unknown and has to be determined from the data. In this section we present two methods for testing whether the successive software runs are independent, or if an output of a run is affected by one or more of its immediately preceding runs [1]. The first method is based on comparing the observed and expected frequencies of different sequences and performing a χ^2 goodness-of-fit test, while the second method is based on the information theory.

A. Tests based on χ^2 goodness-of-fit

The first step is to test the data for independence. For that purpose we compute the frequency of different pairs of runs. If the successive runs are independent, the expected number of times outcome i is followed by outcome j in total of N_2 pairs is given by

$$e_{ij} = N_2 P(i,j) = N_2 P(i)P(j). \quad (6)$$

In order to compare this with observed frequency n_{ij} we estimate the values of $P(i)$ and $P(j)$ as $\hat{P}(i) = n_{i.}/N_2$ and $\hat{P}(j) = n.j/N_2$, which when substituted in (6) leads to

$$\hat{e}_{ij} = n_{i.}n_{.j}/N_2. \quad (7)$$

The estimated values of e_{ij} are then compared with the observed frequencies n_{ij} by means of χ^2 goodness-of-fit test. If the assumption of independence is correct then

$$\chi^2 = \sum_{i,j} \frac{(n_{ij}-\hat{e}_{ij})^2}{\hat{e}_{ij}} \quad (8)$$

is approximately a χ^2 random variable with one degree of freedom[1]. Two comments are in order here. The approximation of (8) with χ^2 distribution is better for larger sample sizes. In addition, for the approximation to apply the values of the expected frequencies should be reasonably large. In judging 'reasonably large' it has been suggested to use the same criteria as in the case of testing independence in a contingency table: less than 20% of the expected values should be smaller than 5 and none should be less than 1 [1].

If the observed value of the test statistics is not significantly large then there is a good agreement between the observed frequencies n_{ij} and expected frequencies $\hat{e}_{ij}$. Specifically, if the value of the statistics (8) is smaller than the critical value of the χ^2 distribution for one degree of freedom and α level of significance (i.e., $\chi^2 < \chi^2_{1;\alpha}$) then the independence assumption cannot be rejected. Otherwise, the independence assumption is rejected and the next step is to test if the first order Markov chain is an adequate model.

The statistical test for the first-order Markov chain is a straightforward generalization of the test of independence; it compares the observed frequencies n_{ijk} of all possible triplets (i.e., $000, 001, 010, \ldots 111$) with the expected frequencies e_{ijk} calculated based on the assumption that the process can accurately be represented by a first order Markov chain.

If successive observations can be accurately represented with a first order Markov chain, the expected number of times outcome i is followed by outcome j which is then followed by outcome k is given by

$$\begin{aligned} e_{ijk} &= \sum_{i,j,k} n_{ijk}P(i,j,k) = \sum_{i,j,k} n_{ijk}P(i,j)P(k|i,j) \\ &= \sum_{i,j,k} n_{ijk}P(i,j)P(k|j). \end{aligned} \quad (9)$$

The probabilities $P(i,j)$ and $P(k|j)$ may be estimated from the triplets n_{ijk} as:

$$\hat{P}(i,j) = n_{ij.}/N_3 \quad (10)$$

$$\hat{P}(k|j) = n_{jk.}/n_{j..} \quad (11)$$

It follows that

$$\hat{e}_{ijk} = n_{ij.}n_{jk.}/n_{j..} \quad (12)$$

[1]In general, the degree of freedom for c different outcomes is $(c-1)^2$.

The goodness-of-fit test statistics

$$\chi^2 = \sum_{i,j,k} \frac{(n_{ijk} - \hat{e}_{ijk})^2}{\hat{e}_{ijk}} \tag{13}$$

has been shown to be asymptotically a χ^2 random variable with $c(c-1)^2$ degrees of freedom if a first order Markov chain is appropriate. In our case, $c = 2$ so the χ^2 distribution has two degrees of freedom.

The same procedure can be used to test for higher order dependencies. It should be noted, however, that the χ^2 approximation gets poorer as the order increases. The reason behind this is the fact that the number of possible sequences of length p given by c^p increases and many of them may have zero frequencies, even for a quite large sample size. However, when the number of outcomes is small (two as in our case or possibly three), the χ^2 goodness-of-fit test can be used up to about third order dependency or even higher, depending on the amount of data available [1].

When the χ^2 approximation is invalid one can look at a few observed sequences and compare them by eye with the theoretical expected frequencies. An alternative, more useful approach is to use the information theory as described next.

B. Tests based on information theory

The amount of information associated with an event which has probability p can be measured by $\log(1/p) = -\log p$. (The logarithmic base is 2.) With c outcomes, having respective probabilities $P(i)$, the average amount of information, often called entropy or average uncertainty, is given by

$$H = E[-\log p] = -\sum_i P(i) \log P(i). \tag{14}$$

The maximum value of H is $\log c$ (in our case $\log 2 = 1$) and this occurs when the outcomes are equally probable for all i. The estimate $\hat{H}_1$ is obtained by substituting the estimate $\hat{P}(i) = n_i/N_1$ in equation (14).

In order to test whether the successive events are independent, $\hat{H}_1$ may be compared with the estimate of the average uncertainty for pairs of software runs, which is given by

$$H(\text{pairs}) = -\sum_{i,j} P(i,j) \log P(i,j) \tag{15}$$

If the successive runs are independent then $H(\text{pairs}) = 2H_1$. Otherwise, $H_1 < H(\text{pairs}) < 2H_1$. It appears that it is the difference between $H(\text{pairs})$ and H_1 which measures the conditional uncertainty about an event given the preceding event:

$$H_2 = H(\text{pairs}) - H_1. \tag{16}$$

The estimate $\hat{H}(\text{pairs})$ is obtained using $\hat{P}(i,j) = n_{ij}/N_2$. Since it is unadvisable to mix frequencies obtained from sequences of different length, for the estimate of H_1 in equation (16) the estimate $\hat{P}(i) = n_{i.}/N_2$ is substituted in (14).

Similar procedure is followed for higher order dependencies. For example $H_3 = H(\text{triples}) - H(\text{pairs})$. The sequence $H_0 = \log c, H_1, H_2, \ldots$ is a decreasing sequence which measures the conditional uncertainty for each order of dependence. The difference between the successive values of H_i, given by $T_i = H_i - H_{i+1}$, measures the amount of information gained by basing the predictions on the previous i runs rather than on the previous $(i-1)$ runs. Thus, at each stage the question arises as to whether

$$\hat{T}_i = \hat{H}_i - \hat{H}_{i+1} \tag{17}$$

is significantly large. In fact, if $\hat{H}_i$ and $\hat{H}_{i+1}$ are both estimated from the frequencies of sequences of length $(i+1)$, then it can be shown that the statistics

$$2 \cdot \ln 2 \cdot N_{i+1} \cdot \hat{T}_i \tag{18}$$

is the same as the likelihood ratio test statistics for testing the null hypothesis that the sequence of events is an $(i-1)$th order Markov chain. Thus, the information theory statistics (18) is asymptotically χ^2 distributed with the following degrees of freedom:

i	Degrees of freedom
0	$c-1$
1	$(c-1)^2$
≥ 2	$c^{i-1}(c-1)^2$

(19)

This also means that we can expect a significance test on $\hat{T}_i$ to have the same disadvantages as the χ^2 goodness-of-fit test with respect to the χ^2 approximation not being valid for higher order Markov chains, especially if the number of outcomes is large and/or the sample size is small.

However, even when the χ^2 approximation is not valid, the values of conditional uncertainty $\hat{H}_i$ lead to a greater understanding of the problem and provide a valuable visual aid in determining the appropriate model for the observed sequence. When the values of $\hat{H}_i$ are plotted against i it is often possible to see the point at which $\hat{H}_i$ starts to decrease relatively slowly and hence to ascertain the order of dependency. In that regard, the graphical approach based on the sequence of conditional uncertainties $\hat{H}_i$ is complementary to and often more reliable than a series of significance tests based on the χ^2 approximation.

V. Experimental setup and case studies

The experimental setup used for our research is presented in Figure 3. The baseline experiment is to run the test cases in a order given in the original regression test suite or other available test pools (the left branch in Figure 3).

One of our goals is to test whether the order of running the test cases affects the outcome of each individual run (i.e., pass and fail), which typically is due to the existence of an internal state that (in addition to the input) affects the output of a software execution. Therefore, we first parse the

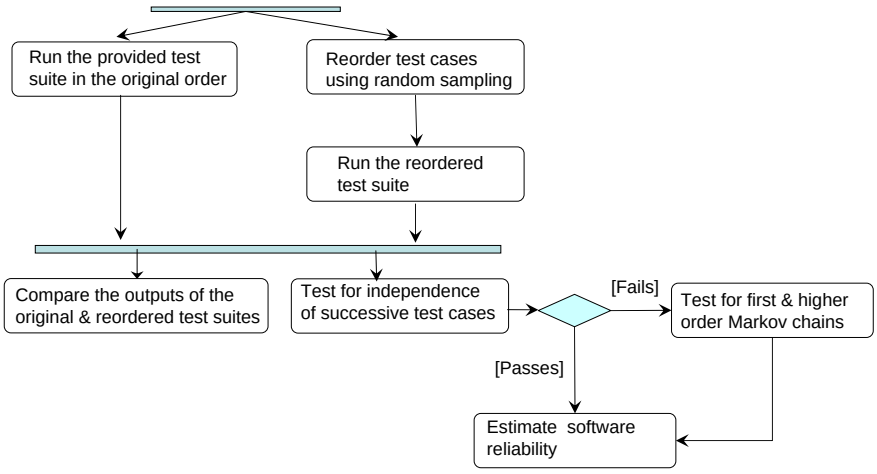

Figure 3. Experimental setup

original test suite and extract the individual test cases. Next, we run 30 test suites, each with the test cases rearranged in a different random order (the right branch in Figure 3). Any change in the outcome (i.e. pass in original test suite becomes a failure in the reordered test suite) shows that the outcome depends on the order of execution due to an internal state which is not reset between test runs.

For the test suite with the original order of the test cases and randomly reordered test suites we use the statistical tests for the order of the Markov model described in section IV. Finally, after selecting the appropriate model, we estimate the probability of failure (i.e., the software reliability).

It should be noted that the above described process was automated to the largest possible extent.

A. Description of the case studies

For empirical evaluation of the assumption of independence and studying the effect of dependent software runs on failure probability (i.e., software reliability) we use four case studies based on three real software applications: Indent [15], GCC [14] and Space [16].

We chose the open source applications Indent and GCC since regression test suites are available with each release of these applications, including drivers to automatically run the test suites and checkers that compare the outputs of the test cases to the expected results. The drivers and the checkers play the role of a test oracle in our studies.

The experimental setup for the open source programs Indent and GCC was similar – we used the regression test suite of a newer version to test an older version, which enables more failures to happen. Special care was taken to exclude test cases designed to test features implemented in newer versions. For both applications we ran the test cases in the order given by the developers. The basic facts of Indent and GCC are as follows:

- Indent is a C code beautifier which changes the appearance of C programs [15]. The release considered in this paper (2.2.0) has around 11,000 lines of code (with around 5,900 lines of non-comment code). We ran the 158 test cases available in the regression test suite of Indent version 2.2.9 on version 2.2.0. After removing the test cases not intended for testing the version 2.2.0 we were left with 152 test cases in the regression test suite, out of which 27 failed.
- GCC C compiler is a part of the GNU Compiler Collection [14]. The release considered in this paper (3.2.3) has over 300,000 lines of code. We ran the regression test suite designed to test the C pre-processor (CPP) part of GCC release 3.3.3 to test GCC 3.2.3. After removing the test cases not intended for testing the version 3.2.3 the regression test suite consisted of 2,424 test cases, out of which 158 failed.

Our third case study, Space, is an interpreter for an array definition language (ADL), used within a large aerospace application. The source code and test cases for Space were downloaded from [16]. The program consists of over 6200 lines of non-comment code. There are 33 versions of Space available at [16], each containing a single fault which was discovered during development and testing. The test suite for Space available at [16] was constructed in two stages. The first 10,000 test cases were randomly generating test cases for the study [13]. Additional test cases were added by the authors of [9] until each executable statement or edge in the control-flow graph was exercised by at least 30 test cases. The total number of test cases used in our study is 13,555[2]. In this study we experimented with two versions: Space A with one fault and Space B with five faults. We ran

[2]We have omitted the last 30 test cases which all seem to be testing what happens when given one parameter does not name an existing file.

the same test suite of 13,555 test cases in the order given at [16] on both Space A and Space B vesrions. The number of failed test cases was 709 for Space A and 2,369 for Space B.

It should be noted that the regression test suites of Indent and GCC or the test suite available for Space represent one possible operational profile, which is not necessarily representative of the real usage of these programs. This fact, however, does not limit the validity of our results since our goal is to illustrate and validate empirically the theory for assessing software reliability in case when successive software runs are dependent, rather than to estimate the reliability as seen by the users.

VI. Empirical results

The experimental setup proved that none of our case studies has an internal state that can, in addition to the input, affect the outcome (i.e., pass or fail) of a software run. Thus, the outcome of each individual run was the same, regardless of the order in which test cases were run. Furthermore, the hypothesis that the successive software runs are independent cannot be rejected at significance level of 0.05 for the randomly reordered test suites for all case studies.

Next, we present the detailed empirical results for the four case studies, with sequences of test cases ran in the original order provided by the developers or other researchers.

A. Analysis of Indent

We start our presentation of the results with Indent. The value of χ^2 statistics for the test of independence given by equation (8) is 11.70140. Since this value is greater than the critical value $\chi^2_{1;0.05} = 3.84146$ the null hypothesis of independent software runs is rejected.

Based on the frequencies of sequences with length three, and using the equation (13) we get $\chi^2 = 0.17504$ which is smaller than the critical value $\chi^2_{2;0.05} = 5.99146$. This means that the null hypothesis that the sequence of software runs can be modeled with a first order Markov chain cannot be rejected at significance level of 0.05, that is, the model shown in Figure 1 is an appropriate model for the sequence of test cases in the regression test suite of Indent.

The same conclusion can be reached based on the information theory approach. Based on the frequencies of the sequences with length three, the estimates of H_i, $\hat{T}_i = \hat{H}_i - \hat{H}_{i+1}$, statistics $2 \ln 2 \cdot N_{i+1}\hat{T}_i$, and the critical value $\chi^2_{df;0.05}$ for degrees of freedom (df) given in (19) are given in Table I. Since the value of the statistics $2 \ln 2 \cdot N_3\hat{T}_2 = 0.17467$ is smaller than the critical value $\chi^2_{2;0.05} = 5.99146$ the hypothesis that the sequence of software runs can be modeled by a first order Markov chain cannot be rejected. Note that H_4 cannot be estimated since the sequence 1101 has not been observed (i.e., its corresponding probability is 0), which means that $\hat{T}_3 = \hat{H}_3 - \hat{H}_4$ and the corresponding statistics cannot be computed. These are annotated with '–' in the Table I.

As it can be seen in Figure 4 which plots the values of the conditional uncertainty H_i for $i = 1, 2, 3$, there is a big drop from $\hat{H}_1$ to $\hat{H}_2$ indicating that the average conditional uncertainty when the previous outcome is known is relatively small, that is, the first order Markov chain models well the sequence of Indent runs which confirms the results of the χ^2 tests.

i	$\hat{H}_i$	$\hat{T}_i = \hat{H}_i - \hat{H}_{i+1}$	$2 \ln 2 \cdot N_{i+1}\hat{T}_i$	$\chi^2_{df;0.05}$
1	0.68008	0.04727	9.89578	3.84146
2	0.63280	0.00084	0.17467	5.99146
3	0.63196	–	–	–

Table I
TESTS BASED ON INFORMATION THEORY FOR INDENT

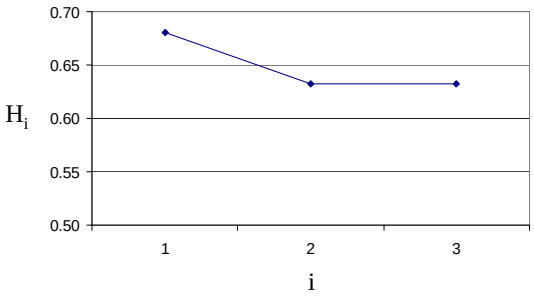

Figure 4. Conditional uncertainty of Indent

Next, we estimate the parameters of the first order Markov chain and discuss the effects of dependent software runs on estimate of software reliability. The estimate of the transition probabilities, again based on sequences of length three, are given by $\hat{P}(j|i) = \hat{p}_{ij} = n_{ij.}/n_{i..}$. The corresponding transition probability matrix P given by (1) then becomes

$$P = \begin{bmatrix} 0.86992 & 0.13008 \\ 0.59259 & 0.40741 \end{bmatrix}. \qquad (20)$$

As discussed in section III-B since $p_{11} = 0.40741 > p_{01} = 0.13008$ the runs are positively correlated. Thus, the probability of failure when the previous run has failed $p_{11} = 0.40741$ is over three times larger than the probability of failure when the previous run has succeeded $p_{01} = 0.13008$. We estimate the unconditional probability of failure per execution $\theta = 0.18000$ using equation (3). The fact that $p_{11} > \theta$ (i.e., $0.40741 > 0.18000$) shows that failures tend to cluster, i.e., the probability of a failure given a failure occurred in the previous execution p_{11} is 2.26 times greater than the unconditional failure probability θ. This observation is very important when the goal is to predict one or multiple step probability of observing a given outcome (i.e., pass or fail) of a software run.

Further, the independent model which leads to $\theta_{\text{ind}} = 0.17763$ underestimates the failure probability compared to the result provided from the dependent model $\theta = 0.18000$,

that is, *the independent model provides an optimistic estimate of the software reliability*.

B. Analysis of GCC

Using the tests based on comparing the observed and expected frequencies of sequences in case of GCC rejects the independent model, as well as the first and second order Markov chain models since, as it can be seen from Table II, in each of these cases the estimate of χ^2 statistics is greater than the corresponding critical value at 0.05 significance level $\chi^2_{df;0.05}$.

The null hypothesis that the sequence of software runs can be modeled by a third order Markov chain cannot be rejected based on the results presented in the last row in Table II. It should be noted, however, that 12 out of 32 expected frequencies are smaller that 5, which exceeds the 20% recommendation given in [1]. In this case, as in general, the χ^2 approximation may not be valid when the values of expected frequencies are not reasonably large, which often happens when testing for higher order dependency.

This case study is a good illustration for using the information theory approach to support the hypothesis that a higher order Markov chain is a good model for a sequence of software runs. Based on the estimates given in Table III the hypotheses that the independent model and the first and second order Markov chains are adequate models are rejected. This approach does not allow us to test the third order model since H_5 cannot be estimated due to the fact that the probability of observing the sequence 11101 is zero (i.e., no such sequence has been observed in our sample).

However, when the values for H_i are plotted against i one can notice that there is a big drop from $\hat{H}_1$ to $\hat{H}_2$, as well as from $\hat{H}_2$ to $\hat{H}_3$. The drop from $\hat{H}_3$ to $\hat{H}_4$ is one magnitude lower which indicates that the increase of the order beyond the third order is unlikely to have any practical significance. Thus, the graphical approach based on the information theory also supports the third order model.

Based on the sequences of length five, the transition probabilities $p_{ijkl} = P(l|ijk)$ are estimated as $\hat{p}_{ijkl} = n_{ijkl.}/n_{ijk..}$, which leads to the transition probability matrix given by (21). The unconditional probability of failure at any software run $\theta = P(001) + P(011) + P(101) + P(111) = 0.06413$, is slightly lower that $\theta_{\text{ind}} = 0.06518$. It appears that in case of GCC the independence assumption leads to underestimating the software reliability. The reason behind this phenomenon is the fact that the conditional probability of a passing run $P(0|ijk)$ is greater than the conditional probability of a failure $P(1|ijk)$ (i.e., $p_{ijk0} > p_{ijk1}$) for all sequences ijk except for the sequence 111 (see the transition probability matrix (21)).

C. Analysis of Space A

Space A is a version of Space program with one fault (number 24). Based on the comparison of the observed and expected frequencies $\chi^2 = 0.02501$, which is smaller than the critical value $\chi^2_{1;0.05} = 3.84146$. This means that at 0.05 significance level we cannot reject the hypothesis that the sequence of runs of Space A is independent.

Model	χ^2	$\chi^2_{df;0.05}$
Independent	213.98101	3.84146
First order	48.86036	5.99146
Second order	16.14053	9.48773
Third order	10.08294	15.5073

Table II
RESULTS BASED ON χ^2 TESTS FOR GCC

i	$\hat{H}_i$	$\hat{T}_i = \hat{H}_i - \hat{H}_{i+1}$	$2\ln 2 \cdot N_{i+1}\hat{T}_i$	$\chi^2_{df;0.05}$
1	0.34799	0.03792	127.37602	3.84146
2	0.31007	0.01161	38.97881	5.99146
3	0.29846	0.00474	15.92001	9.48773
4	0.29372	–	–	–

Table III
TESTS BASED ON INFORMATION THEORY FOR GCC

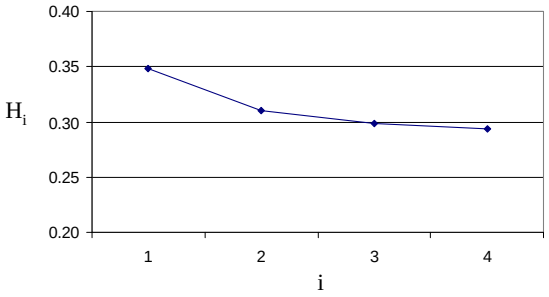

Figure 5. Conditional uncertainty of GCC

The same result is obtained using the information theory approach. Thus, the value of the statistics $2\ln 2 \cdot N_2 T_1 = 0.02483$ is smaller than the critical value $\chi^2_{1;0.05} = 3.84146$, which means that null hypothesis that the sequence of runs is independent cannot be rejected. Even more, as it can be seen in Figure 6 the values of the conditional uncertainty H_i are very close, which basically means that the average conditional uncertainty does not decrease with the knowledge of one or more preceding events.

The likely reason for this result is the fact that the first 10,000 test cases in the testing suite of Space are randomly distributed and 654 of total 709 failures are associated with these 10,000 test cases. In addition, very few of the remaining 55 failures associated with the last 3,555 test cases occur in clusters (i.e., close to each other) which leads to observed frequencies very close to the expected frequencies under the assumption of independence. The estimate of the unconditional failure probability of Space A is $\theta_{\text{ind}} = 0.05231$.

D. Analysis of Space B

Next we present the analysis of the results of a version of Space with five faults (number 3, 8, 14, 23, and 33). The results of the χ^2 test based on the comparison of the observed and expected frequencies of sequences are given

$$P = \begin{bmatrix} 0.96284 & 0.03716 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.78161 & 0.21839 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.85000 & 0.15000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.56522 & 0.43478 \\ 0.88636 & 0.11364 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.75000 & 0.25000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.86957 & 0.13043 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.32258 & 0.67742 \end{bmatrix}. \quad (21)$$

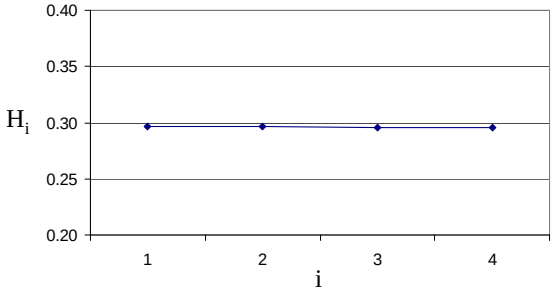

Figure 6. Conditional uncertainty of Space A

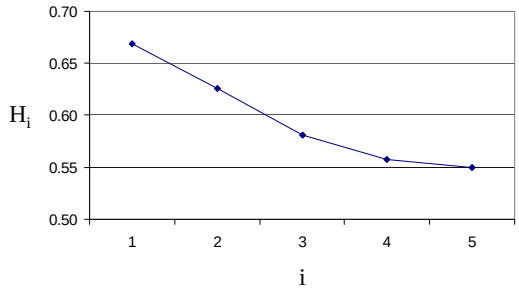

Figure 7. Conditional uncertainty of Space B

Model	χ^2	$\chi^2_{df;0.05}$
Independent	936.75542	3.84146
First order	815.92354	5.99146
Second order	462.18663	9.48773
Third order	258.16469	15.50730

Table IV
RESULTS OF THE χ^2 TESTS FOR SPACE B

in Table IV. Obviously, the hypotheses that the sequence of Space B runs is independent or can be modeled with up to the third order Markov chain are all rejected since the estimates of the χ^2 statistics are much higher than the corresponding critical values.

The same observation is made using the approach based on information theory; the estimates of the statistics $2 \ln 2 \cdot N_{i+1} \hat{T}_i$ for $i = 1, 2, 3, 4$ are all much higher than the corresponding critical values of χ^2 distribution with the appropriate degrees of freedom at significance level 0.05. Further, as it can be seen from the plot of the conditional uncertainty H_i shown in Figure 7 there is a noticeable drop even from $\hat{H}_4$ to $\hat{H}_5$. We did not run additional tests for higher order Markov chains since the accuracy of the χ^2 approximation of both test statistics decreases with the order of the chain and it is considered that the χ^2 goodness-of-fit test can be used up to about third order dependency, even in case when the number of possible outcomes is small [1].

Space B is an example of a case study for which Markov chain model of any order may not be an appropriate model. In this specific case the reason for this result is the way Space test suite was created, with the first 10,000 tests randomly selected and the remaining 3,555 test cases added until each executable statement or edge was exercised by at least 30 test cases. In particular 1,510 out of the total 2,369 failures of Space B were distributed among the first 10,000 test cases. To test our hypothesis that the results are due to the way Space test cases were arranged in [16], we ran the tests for the order of the Markov chain model on the subset of the first 10,000 randomly generated test cases. The value of χ^2 based on the comparison of the observed and expected frequencies is 0.02354, which is smaller than the critical value $\chi^2_{1;0.05} = 3.84146$, leading to conclusion that at the significance level of 0.05 the assumption of independence cannot be rejected. The result was confirmed by the test based on the information theory and the graph of the conditional uncertainty. This means that the successive software runs of Space B consisting of the first 10,000 test cases are independent, with unconditional failure probability $\theta_{\text{ind}} = 0.15100$.

In other words, the reason the independent and Markov models up to the third order failed on the whole test pool of 13,555 test cases (see Table IV) was due to the way the reminding failures (i.e., 859) were distributed among the 3,555 additional test cases. Namely, we identified a large number of uninterrupted sequences of failures, likely due to the fact that these 3,555 test cases were added to improve the coverage of some statements and edges and thus in large numbers were triggering same faults. As expected, this type of behavior could not be modeled as in independent sequence or Markov model, which was confirmed by the fact that, when run on the sequence of outputs from 3,555 test cases, the statistical tests rejected the hypothesis of independence and up to the third order Markov chain, similarly to the results presented in Table IV.

VII. Conclusion

In this paper we have addressed the estimation of probability of failure (i.e., software reliability) when successive software runs are dependent. The contributions include generalization of the previous work to accommodate higher order dependencies and an empirical study based on three real software applications. We used two different approaches for testing the order of dependency, one based on the difference between the observed and expected frequencies of sequences and the other based on the information theory. It appeared that the conditional uncertainty, when plotted for different possible orders of dependency, can be used as a complementary approach to the two statistical tests, which is especially useful when testing higher order models for which the χ^2 approximation may not be valid.

A brief summary of the empirical results is as follows:

- The test of independence failed for both open source case studies: Indent and GCC. The appropriate models for these two applications are the first and third order Markov chains, respectively. The good fit likely is due to the fact that in regression test suites similar test cases often tend to occur close together, which tend to result in a series of related software runs. It is interesting to note that the independence assumption in case of Indent led to underestimating the unconditional failure probability, while in case of GCC in overestimating the unconditional failure probability. In either case, the conditional probabilities of failure differed significantly depending on the immediately preceding run in case of Indent, that is, on the three preceding runs in case of GCC. This observation is very important, especially if the goal is to use the model to predict one or multiple step probability of observing a given outcome (i.e., pass or fail) of a software run.
- The independence of successive runs was not rejected for Space A case study, which basically has a small number of failures, mainly distributed among the part of the test pool consisting of a large number of randomly selected test cases. On the other side, the independence and Markov chain models up to the third order were rejected for the Space B case study. Detailed analysis of the sequence of outputs showed that this results were due to the way test cases were generated and arranged in the test pool. The statistical tests on the first part of the test pool which consisted of randomly generated test cases showed a good fit with the independence model. This observation leads to conclusion that researchers and practitioners should thoroughly explore the data when conducting empirical studies.

There may be case studies for which Markov chain model of any order may not be appropriate. Considering other more complex models of dependency is a topic of our future work.

Acknowledgments

We thank Arin Zahalka and Jeffrey Zemerick for their help with data collection. This work was supported by NASA OSMA SARP grant managed through NASA IV&V Facility and by the NSF grants CNS-0447715 and CCF-0916284.

References

[1] C. Chatfield, "Statistical Inference Regarding Markov Chain Models", *Applied Statistics*, Vol.22, 1973, pp. 7-20.

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

[3] Y-S. Dai, M. Xie and K-L. Poh, "Modeling and Analysis of Correlated Software Failures of Multiple Types", *IEEE Trans. on Reliability*, Vol.54, No.1, Mar. 2005, pp. 100-106.

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

[5] K. Goseva-Popstojanova and K. S. Trivedi, "Failure Correlation in Software Reliability Models", *IEEE Trans. on Reliability*, Vol.49, No.1, Mar. 2000, pp. 37-48.

[6] K. Goseva-Popstojanova, M. Hamill and X. Wang, "Adequacy, Accuracy, Scalability, and Uncertainty of Architecture-based Software Reliability: Lessons Learned from Large Empirical Case Studies", *17th Int'l Symp. on Software Reliability Engineering (ISSRE 2006)*, Nov. 2006, pp. 197-203.

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

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

[9] G. Rothermel, R. H. Untch, C. Chu and M. J. Harrold, "Prioritizing Test Cases for Regression Testing", *IEEE Trans. on Software Engineering*, Vol.27, No.10, 2001, pp. 929-948.

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

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

[12] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, John Wiley & Sons, New York, 2002.

[13] F. I. Vokolps and P. G. Frankl, "Empirical Evaluation of the Textual Differencing Regression Testing Technique", *Int'l Conf on Software Maintenance*, Nov. 1998, pp. 44-53.

[14] http://gcc.gnu.org/

[15] http://www.gnu.org/software/indent/indent.html

[16] http://sir.unl.edu/portal/index.html