STVR - Software Testing, Verification and Reliability, Vol. 9, No. 3, pp. 145-166, John Wiley and Sons Ltd., 1999.

A Contribution to the Evaluation of the Reliability of Iterative-Execution Software[1]

Andrea Bondavalli[1], Silvano Chiaradonna[1], Felicita Di Giandomenico[2] and Lorenzo Strigini[3]

[1] CNUCE-CNR, Via S. Maria 36, 56126 Pisa, Italy

[2] IEI-CNR, Via S. Maria 46, 56126 Pisa, Italy

[3] Centre for Software Reliability, City University, Northampton Square, London EC1V OHB, UK

Abstract

This paper deals with the reliability of software executed iteratively, as e.g. in process control applications. The probability of mission survival is evaluated taking account of two characteristics of iterative software: a) system failure is defined in terms of the behaviour of the software over successive iterations, because the controlled system can usually tolerate short bursts of errors; b) the probabilistic correlation between successive executions of the software, which is to be expected for various reasons. The paper presents models accounting for these characteristics and evaluates their effects. The interesting case of fault-tolerant software is considered as well. Using the example of a "pair-and-spare" type fault-tolerant scheme, the relationships between different aspects of failure behaviour that are covered by the models developed here and those used elsewhere for fault-tolerant software are shown.

Key words: Iterative software, Reliability, Analytical modelling, Correlation between executions, Failure clustering.

1 Introduction

This paper deals with the evaluation of the *probability of completing a mission* of fixed duration (reliability at mission end time) of *software components*, executed iteratively for process-control applications, with periodic scheduling. The intent is to show how different pa-

---

[1] Based on “Dependability Models for Iterative Software Considering Correlation among Successive Inputs,” by Bondavalli, Chiaradonna, Di Giandomenico and Strigini, which appeared in Proc. IEEE IPDS’95, pp. 13-21.

rameters of a system affect its suitability for different applications. The paper offers two contributions:

1 it explores the consequences of two characteristics of control applications with iterative executions:

1.1 system failure must be defined in terms of the behaviour of the software over successive iterations, because the controlled system may be able to tolerate errors of the control software only if they happen in isolation or in short bursts;

1.2 the probabilistic correlation between successive executions of the software is an important factor in determining the failure behaviour of the system. Positive correlation (i.e., a "bursty" failure process) is to be expected for various reasons, not least the fact that the input values representing physical variables of the controlled system evolve along a "trajectory" in the input space of the software.

These effects are studied using "black box" models, where the smallest entity in the models is one complete iteration of software execution. Upper and lower bounds on reliability for this general class of models are also given.

2 it then shows how these effects change the reliability predictions obtained for *fault-tolerant* software in previous literature via "clear box" models. These predicted reliability from parameters describing the inner working of one iteration of a redundant software component. These "clear box" models for one iteration are here substituted for the "black box" in the new models.

Correlation between successive executions has been studied in a few recent papers. [Chen and Mills, 1996, Strigini, 1996] point out the difficulties it creates in estimating reliability from operational testing. More complete studies, for the limited case of fault-tolerant software structures, have been published in [Csenki, 1991] and then in [Tomek, Muppala and Trivedi, 1993]. This paper extends the approach used by these previous studies.

Fault-tolerant software using (identical or diverse) coarse-scale replication has been studied and modelled extensively, since its applications tend to be critical ones. A fault-tolerant software component consists of a set of redundant sub-components (either identical *replicas* or diversely implemented, functionally equivalent *variants*), plus *adjudication* sub-components. Such modular-redundant designs differ in how they organise the execution of their subcomponents ("parallel" vs. "serial", etc.). All these solutions may be described by similar mathematical models, in which varying degrees of diversity are represented by different values of model parameters. Evaluating the reliability of these systems is made difficult by the impossibility of assuming statistical independence between the failures of different variants ([Brilliant, Knight and Leveson, 1990, Knight and Leveson, 1986, Littlewood and Miller, 1989]). Numerous papers

(most recently [Arlat, Kanoun and Laprie, 1990, Chiaradonna, Bondavalli and Strigini, 1994, IEEE-TR, 1993, Laprie, et al., 1990, Tai, Meyer and Avizienis, 1996]) deal with modelling the effectiveness of software fault tolerance through structural ("clear box") models, all assuming independence between successive executions.

The goal of this paper is to explore the effects of the factors listed above in bullet points 1.1 and 1.2, generally ignored in the previous literature, studying which aspects and descriptive parameters would be important in determining these effects, and thus providing some insight and support for practitioners. It will appear from the discussion that evaluating an individual new product remains a quite formidable task. However, rigorous modelling allows one to understand the effects of different parameters, with an intuitive real-world meaning, on reliability. It is thus a protection against drawing unwarranted conclusions about a new system, and can inform the choice of development methods by indicating which parameters these should tend to increase or to decrease. Furthermore, models like these could be used for (probabilistic) predictions about a new product if indications on the distributions of the parameters can be obtained from populations of similar products already in operation.

The paper is organised as follows. Section 2 defines the systems considered and the problems under study. Section 3 introduces the "black box" models of sequences of executions, starting from the simple model with independence and introducing one new factor at a time. Section 4 studies the effects of the distribution of failure burst lengths. Section 5 completes the comparison between the developed (most complete) model and the independence-based model, with a simple example in which the "burstiness" of failures can be described by a single parameter. In Section 6, an example of "clear box" model of fault-tolerant software is recalled and combined with the previous models allowing for a bursty failure process and robust controlled systems. Section 7 summarises and discusses the results presented.

2 The system under study

The characteristics of the systems under study are first described. Then, correlation between iterations and effects of failure bursts, which change the failure model to consider, are briefly discussed.

2.1 Iterations, missions and failures

An application of an iterative nature is assumed: a *mission* consists of a series of *iterations* of the execution of the software. At each iteration, the software accepts an input and produces an output. The outcomes of an individual iteration may be: i) a *correct result*, ii) a *benign failure* (i.e., a failure which, in isolation, can be tolerated by the controlled system, and does not abort the mission) or iii) a mission failure (either a single computer failure that would abort the mis-

sion - often called a *catastrophic failure* of the computer - or a pattern of individually benign failures such that the controlled system cannot tolerate it).

Benign failures can be detected during testing of the software as deviations from the software's intended behaviour, which, when analysed, are found not to be dangerous for the controlled system. In actual operation in the field they may be detected (though this is not certain) at the software level (e.g. as disagreements in voted redundant systems) or at the system level, as the controlled system departs from the behaviour intended by the software designers but remains within the acceptable envelope specified in the high-level requirements.

From the point of view of software, a different classification is appropriate: each execution can result in: i) *success*, i.e., the delivery of a correct result, ii) a *detected error*, i.e., the execution is stopped by checks internal to the software or by a watchdog timer, and no result is delivered, or iii) an *undetected error* (delivery of an erroneous result). In this study, all detected failures are assumed to be benign, and all and only the undetected failures are assumed to be catastrophic. This is reasonable in some application scenarios, e.g., a protection system which may trigger the safe shut-down of a controlled process both if it detects a dangerous process situation and if it detects an internal error. The timing of executions is assumed to be a static, periodic scheduling of iterations, typical of control or safety systems. Thus a discrete time scale, with one time unit equal to the period of scheduling of the software can be used. At the end of each period, watchdog timers abort any components that are still running; should this situation occur, a (detected) timing failure is assumed.

2.2 Correlated executions and new failure models

Many models of iterative software assume independence between the outcomes of successive iterations, which rarely holds in reality. In control systems, the inputs usually follow a trajectory of small steps in the input space, representing points in the state space of the controlled object. Experiments have shown contiguous failure regions in the program input space, i.e., connected sets of input points on which the program will fail. It has also been argued on theoretical grounds that this clustering of failures is generally to be expected [Bishop, 1993]. These arguments do not actually require the existence of connected failure regions, but simply of regions where failure points are much more frequent than they are on average on the whole input space. Then, failures of the software must usually be grouped in *clusters* [Amman and Knight, 1988, Bishop and Pullen, 1988], rather than being isolated events. When the input trajectory intersects a "failure region" in the input space, a failure burst occurs. Then, an error at one execution indicates a high probability (i.e., higher than the marginal probability for that software) of an error at the next execution. The likely number of failures in a series depends on the size of the failure region (measured along the trajectory), and the "speed" of the trajectory (i.e., the

statistics of the distance between two subsequent inputs). In the rest of this discussion only the combined effect of these two factors, i.e., the distribution of the number of iterations during one crossing of a failure region, will be taken into account. This distribution would in practice be either measured experimentally (in which case, however, the experiment could allow one to measure reliability directly), or guessed by the evaluator using knowledge about the input space and the "speed" with which a trajectory crosses its different regions. By only considering the number of steps inside a failure region, the length of a step in the trajectory (distance between two inputs) is used in practice as the unit of measurement for distances in the input space. There may be applications which do not allow an intuitively sound definition of a distance function over the input space, or of a trajectory through it, but a model formulated in terms of the number of steps in a failure region is still applicable.

The possibility that long bursts of benign failures (as may be expected with a high positive correlation between successive iterations) contribute to mission failures must also be modelled. This is done by assuming that the controlled system can survive individual benign failures of the control computer, or short sequences thereof, but any series of n_C or more benign failures in a row will cause the mission to fail (the parameter n_C is called here "robustness factor" of the controlled system). This is a common characteristic of continuous-control systems. Some controlled systems have enough physical inertia that an individual erroneous output from the control system cannot move them into a prohibited state. The probability of an isolated failure producing catastrophic consequences is thus zero. In other cases, some control system failures are immediately catastrophic, but others (e.g., those where the control system internally detects its own failure and outputs a safe value to the controlled system) are not: only if they are repeated the resulting lack of active control may cause the controlled system to drift into a dangerous state. In either case, assuming that any sequence of up to n_C-1 failures is tolerated and all longer sequences are catastrophic is still a simplification of reality, yet more realistic than assuming that a controlled system can tolerate any arbitrary series of benign failures.

3 "Black box" modelling of sequences of executions

This section first defines the assumptions made for modelling correlation between successive iterations and then shows the effects of different sets of modelling assumptions, introducing them one at a time.

3.1 "Black box" model for correlation

The model of correlation between successive iterations is derived from those used in [Csenki, 1991, Tomek, et al., 1993]. There are two kinds of failure events for a software component:

i) point failure: which happens when the input trajectory enters a failure region,

ii) serial failure: a number of consecutive failures, happening with probability 1 after the occurrence of a point failure, i.e., after the input trajectory enters a failure region. The number of serial failures subsequent to any point failure is a random variable, its distribution can be chosen to represent what is believed to be the stochastic characteristics of the input trajectories and the failure regions.

In [Csenki, 1991], this model is applied to the primary variant in a recovery block scheme. A simple discrete-time Markov chain allows an analytical evaluation of the reliability (MTTF) of recovery blocks. [Tomek, et al., 1993] analysed the different forms of correlation in the recovery block structure, including correlation among the different alternates and among alternates and the acceptance test on the same inputs. For the correlation between successive inputs, [Tomek, et al., 1993] uses the same assumptions as [Csenki, 1991], including the same event set, and evaluates its effect on the MTTF via a SRN (Stochastic Reward Nets) model.

To this model, two variations are added:

- any failure of the software (either "point" or "serial") may be either benign or catastrophic (immediate mission failure) for the controlled system, as defined in Section 2.1;
- all sequences of at least n_C benign failures cause mission failure.

Failure Regions: *A priori*, the probability that the input to an iteration belongs to a failure region is assumed to be the same for every iteration. One could assume a different distribution if it were known, for instance, that some parts of the input space for a specific system are more prone to failures than others. Once the trajectory enters a failure region, though, the probability that the next input will cause a failure (i.e., that it will still belong to the failure region) increases. In [Bishop, 1993] some two-dimensional views of fault regions are shown for a specific program, and a number of factors affecting the shapes of the faults were identified. There is as yet no empirical basis for assuming some specific shape to be the general case. Here, some shapes are chosen which are simple to model, and bounds are evaluated which do not depend on the specific assumptions which are difficult to verify.

Input Sequence: The inputs form a "trajectory", that is, every input value is close to the previous one. A random or deterministic walk trajectory is assumed. The step length must be small compared to the size of the input space: if the step length becomes comparable to the size -in each dimension- of the input space, inputs tend to being uniformly and independently distributed [Bishop and Pullen, 1988]. Many different types of trajectories may be considered. Examples are: 1) the next input is obtained from the previous one by modifying the values on each dimension by a small, random quantity; 2) (a subcase of 1) a "forward-biased" trajectory: passing from one input to the next the direction may only change slightly; 3) (a subcase of 2) a trajectory of points on a straight line, at a random, small distance from each other.

Such a model is appropriate for a situation in which the following assumptions are true:

1) a single success before the n_c-th consecutive failure will bring the system back to a stable state, i.e. the memory of the previous failure sequence is immediately lost;
2) the trajectory of the input sequence is "forward-biased": passing from one input to the next the direction may only change slightly;
3) the failure regions are convex.

These assumptions are used to simplify the modelling without restricting too much the class of applications that can be modelled. Many control applications (e.g., navigation systems) show "forward-biased" trajectories. Assumptions 2) and 3) imply trajectories that, once they have left a failure region, are unlikely to re-enter it soon. It is thus possible to assume a constant probability of entering a failure region since, after leaving a certain failure region, a) there is a low probability of re-entering it within a small number of iterations and b) after an appropriate number of iterations, the probability of re-entering that failure region is equal to the probability of entering any other failure region.

Only one of the possible causes of correlation between successive outcomes, i.e., continuous failure regions in the input space, is being modelled in the present work. There are other possible causes, e.g., the fact that an error at one execution is likely to corrupt the state of the software and make an error more likely at each subsequent execution. This effect could, in some cases, be satisfactorily represented by these models: the internal state variables can be considered as inputs, and it is plausible that at least some of their values evolve along continuous trajectories. However, further study is necessary.

3.2 Independence between iterations; repeated benign failures cannot cause mission failure

A mission consists of a sequence of n iterations, and the outcomes of the individual iterations (an outcome may be success, benign failure, catastrophic failure, with probabilities p_s, p_b, and $p_c=1-p_s-p_b$, respectively) are independent events.

The probability of completing the mission is that of a series of n executions without catastrophic failure, $(1-p_c)^n$.

This expression of the mission survival probability is probably very pessimistic compared to what one would obtain by assuming a realistic, positive correlation. In fact, if the failures at all executions were completely correlated (either no execution fails or all fail), the same probability would become $(1-p_c)$, no matter how long the mission is.

3.3 Independence between iterations; repeated benign failures may cause mission failure

The assumption that any series of n_c or more consecutive benign failures causes mission failure reduces the probability of surviving a mission (all model parameters being equal). The probability of mission failure if series of benign failures are of no concern, $(1-(1-p_c)^n)$, is then a *lower bound* on the total probability of mission failure. A reasonably tight test of whether the probability of a mission failure due to a series of benign failures, $p_{c\text{-serial}}$, can be neglected can be obtained as follows. Let UB be the probability that a series of iterations without catastrophic failure is terminated by n_c benign failures. UB is clearly an upper bound on $p_{c\text{-serial}}$ (consider that the first $n-n_c$ events in the series may already contain n_c or more benign failures):

$$p_{c\text{-serial}} \leq \text{UB} = \sum_{i=0}^{n-n_c} (1-p_c)^i p_b^{n_c} = p_b^{n_c} \cdot \frac{1-(1-p_c)^{n+1-n_c}}{p_c}$$

Thus the probability of mission failure belongs to the interval $[(1-(1-p_c)^n), (1-(1-p_c)^n)+\text{UB}]$. If UB is negligible in comparison with $(1-(1-p_c)^n)$, then it is legitimate to neglect $p_{c\text{-serial}}$.

3.4 Correlation between successive iterations; repeated benign failures may cause mission failure

The independence assumption is now abandoned. A scenario is now considered in which a benign failure at an iteration of the program makes it more likely than otherwise that the program will fail at the next iteration as well. However, there are other realistic scenarios which are not addressed in this paper. E.g., if a system is engineered to detect its own internal errors and then issue a safe output and reset, one may well assume that most benign failures are due to this mechanism, and are very likely to be followed by a success.

A Markov model that accounts for correlation between iteration and also for the robustness factor of the system, n_c, is in Figure 1. States S (the initial state) and C represent success and catastrophic failure, respectively. The set of states $\{B_1, ..., B_{nc-1}\}$ represent benign failures (more precisely, those benign failures that belong to bursts short enough not to cause catastrophic failure). Splitting the physical state "benign failure" into many model states is a device for representing arbitrary distributions of the length of benign failure bursts, despite the memory-less property of Markov models. After an iteration with success (S), there is a probability p_{sb} of the next execution producing a benign failure (i.e., of the input trajectory entering a failure region). *After* entering a failure region, the probabilities of staying there for one, two or more iterations are given by the parameters p_1, p_2, etc.. These parameters model the correlation between successive failures, i.e., they specify a distribution function for the random variable "number of consecutive failures", conditional on being inside a failure region. For instance, the transition from S to B_2 represents the event (with probability $p_{sb}p_2$) of the

program's input trajectory entering a failure region which it will cross in two iteration steps. While in the failure region, the program may fail catastrophically (arcs leading to C), but otherwise it can only move from B_2 to B_1. Once in B_1, the input trajectory leaves the failure region, moving to another success, or entering another failure region. To represent a situation with sparse, small failure regions in unknown positions, the arcs issuing from B_1 have the same transition probabilities as those issuing from S: the fact of having just left a failure region gives no information about the likelihood of another failure region being near. This model can represent various degrees of positive correlation between successive iterations, and independence as a limiting case (with $p_1 = 1$ and $p_i = 0$ for all $i>1$).

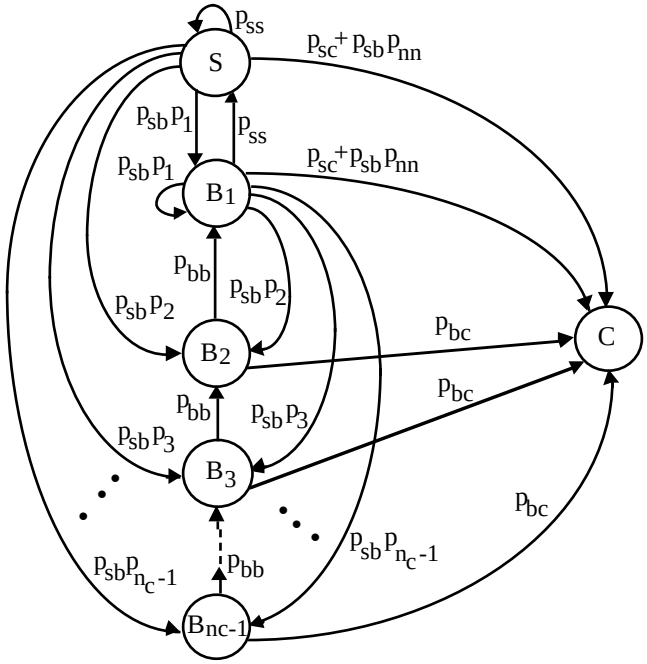

Figure 1. The model for iterative executions with failure clustering and accounting for the robustness factor of the system n_c.

p_{nn} is the probability of a benign failure burst lasting n_c iterations or more, i.e., long enough to cause mission failure: $p_{nn} = 1 - \sum_{i=1}^{n_c-1} p_i$. This path from success to catastrophic failure is described by the term $p_{sb}p_{nn}$ in the expression of the probability of the transition from S to C. Parameters p_{sc} and p_{bc} represent, respectively, the probabilities of immediate catastrophic failure from state S and from any of the states B_i. State C models the failure of a mission due either to crossing a failure region and staying there for at least n_c iterations, or to a catastrophic failure. A third cause for mission failure exists, which has been neglected in the model solution, as the errors thus introduced are negligible [Bondavalli, et al., 1994]: crossing two or more failure regions without interruption, staying in each one for less than n_c failures but exceeding n_c-1 consecutive benign failures.

Under these conditions, the positive correlation between successive iterations will affect reliability both via the probability of having n_c or more consecutive benign failures and via the longer stays in failure regions, with increased risk of catastrophic failure. The following expression applies (where X_i is the outcome of the i-th iteration):

$$P(\text{mission success}) = \prod_{i=1}^{n} P(X_i \neq C \mid X_{i-1} \neq C).$$

Observing that $"X_{i-1} \neq C" = "X_{i-1} = S \text{ or } X_{i-1} = B_1 \text{ or } X_{i-1} \in \{B_2, ..., B_{n_c-1}\}"$, each term in the product can be decomposed as follows.

$$P(X_i \neq C \mid X_{i-1} \neq C) = P\left(X_i \neq C \mid X_{i-1} = S \text{ or } X_{i-1} = B_1 \text{ or } X_{i-1} \in \{B_2, ..., B_{n_c-1}\}\right) =$$

$$= P(X_i \neq C \mid X_{i-1} = S)P(X_{i-1} = S \mid X_{i-1} \neq C) + P(X_i \neq C \mid X_{i-1} = B_1)P(X_{i-1} = B_1 \mid X_{i-1} \neq C) +$$

$$+P\left(X_i \neq C \mid X_{i-1} \in \{B_2, ..., B_{n_c-1}\}\right)P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right).$$

From Figure 1 one can derive $P(X_i \neq C \mid X_{i-1} = S) = 1 - p_{sc} - p_{sb}p_{nn}$,

$P(X_i \neq C \mid X_{i-1} = B_1) = 1 - p_{sc} - p_{sb}p_{nn}$ and $P\left(X_i \neq C \mid X_{i-1} \in \{B_2, ..., B_{n_c-1}\}\right) = 1 - p_{bc}$.

Substituting these probabilities in the previous expression:

$$P(X_i \neq C \mid X_{i-1} \neq C) = (1 - p_{sc} - p_{sb}p_{nn})P(X_{i-1} = S \mid X_{i-1} \neq C) +$$

$$+(1 - p_{sc} - p_{sb}p_{nn})P(X_{i-1} = B_1 \mid X_{i-1} \neq C) + (1 - p_{bc})P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right).$$

Finally, observing that :

$$P(X_{i-1} = B_1 \mid X_{i-1} \neq C) = 1 - P(X_{i-1} = S \mid X_{i-1} \neq C) - P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right)$$

one can derive:

$$P(X_i \neq C \mid X_{i-1} \neq C) = (1 - p_{sc} - p_{sb}p_{nn})P(X_{i-1} = S \mid X_{i-1} \neq C) + (1 - p_{sc} - p_{sb}p_{nn})\left(1 - P(X_{i-1} = S \mid X_{i-1} \neq C) -\right.$$

$$\left.-P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right)\right) + (1 - p_{bc})P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right) =$$

$$= (1 - p_{sc})P(X_{i-1} = S \mid X_{i-1} \neq C) - p_{sb}p_{nn}P(X_{i-1} = S \mid X_{i-1} \neq C) + (1 - p_{sc})\left(1 - P(X_{i-1} = S \mid X_{i-1} \neq C)\right) -$$

$$-(1 - p_{sc})P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right) - p_{sb}p_{nn}\left(1 - P(X_{i-1} = S \mid X_{i-1} \neq C) -\right.$$

$$\left.-P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right)\right) + (1 - p_{bc})P\left(X_{i-1} \in \{B_2, ..., B_{n_c-1}\} \mid X_{i-1} \neq C\right) =$$

then with some algebraic manipulations

$$= (1 - p_{sc}) + (p_{sc} - p_{bc})p\left(X_{i-1} = B_2 \text{ or..or } X_{i-1} = B_{n_c-1} | X_{i-1} \neq C\right) -$$

$$-p_{nn}p_{sb}\left(1 - p\left(X_{i-1} = B_2 \text{ or..or } X_{i-1} = B_{n_c-1} | X_{i-1} \neq C\right)\right)$$

In the right-hand side of this equation, the first additive term would be the only term if one assumed independence and that sequences of benign failures do not affect the probability of mission failures (since in such a case $p_{bc}=p_{sc}$ and $n_c=\infty$). The second term represents the contribution of the higher (or lower, as the case might be) probability of catastrophic failure after a benign failure, compared to that after a success. The third term represents the probability of mission failure due to n_c or more consecutive failures.

4 Effects of different distributions of the length of stay in failure regions

The distributions of i) the duration (in steps) of the crossing of a failure region by a trajectory and ii) the number of steps between two failure regions, determine the reliability for a mission of the software. By assuming failure regions which are small and far apart, the need to model the time between encountering them with any precision can be avoided. The distribution of the stay in the failure regions must be guessed at, by making plausible assumptions derived from one's knowledge of the type of software, and previous experience with similar software. Of course, this cannot produce a precise prediction, but the spread of results obtained with plausible assumptions will be a useful indication.

A useful consideration is that, whatever the chosen distribution, its effect on reliability is determined by two parameters: its mean (which determines the duration of higher exposure to catastrophic failures), and the probability that a sequence of benign failures is equal or longer than the critical threshold n_c. A good estimate of these parameters, even without the knowledge of the complete distribution, should yield a satisfactory prediction. Therefore, the following evaluation will mainly focus on these two factors (while keeping all others constant), by showing the effect of choosing distributions belonging to different families but with the same values of one or the other of these two parameters. The following families of distributions (the first three are common distributions from the literature) have been considered:

- geometric distribution, defined for $q \in (0,1)$ as $p_i = (1-q)q^{i-1},\ i \geq 1$, and for q=0 as $p_1=1$, $p_i=0$ for $i \geq 2$;
- modified negative binomial, defined as $p_i = \binom{i+r-2}{r-1}(1-q)^r q^{i-1},\ i \geq 1$, for $q \in (0,1)$ and $p_1 = 1,\ p_i = 0,\ i \geq 2$ for q=0;
- Poisson, defined as $p_i = \frac{e^{-\alpha}\alpha^{i-1}}{(i-1)!},\ i \geq 1$;
- an "ad-hoc" distribution, as an example of deriving a distribution from assumptions about the microscopic behaviour of the software, for a hypothetical application where the following knowledge has been obtained: i) the input space is a discrete two-dimensional (Cartesian) space; ii) failure regions are squares with their sides parallel to the axes of the input space; iii) the input trajectory is a directed straight line crossing the square failures region vertically, horizontally or diagonally (details are in [Bondavalli, et al., 1994, Bondavalli, et al., 1995]);
- two limiting classes of distributions, which can be shown to provide respectively a lower and an upper bound on the reliability obtainable for given values of n_c and p_{nn}:

- *g* defined such that g(i)=0 for i=1,.., n_{c-2}, $g(n_c - 1) = 1 - p_{nn}$ and $\sum_{i>n_c-1} g(i) = p_{nn}$. , If the input trajectory enters a failure region, it stays in it for at least (n_c - 1) iterations;
- *f* defined such that $f(1) = 1 - p_{nn}$, f(i)=0 for i=2,.., n_{c-1}, and $\sum_{i>n_c-1} f(i) = p_{nn}$. After entering a failure region, the trajectory either exits immediately (after one benign failure) or stays in it for at least n_c iterations, causing a mission failure.

$p_{sb} = 8x10^{-6}$	$p_{bc} = 10^{-4}$ **(notice that**
$p_{sc} = 8x10^{-9}$	$p_{bb} = 1 - p_{bc}$ **$p_{bc} >> p_{sc}$)**
$p_{ss} = 1 - p_{sb} - p_{sc}$	$n_c = 10$
	$n = 10^6$

Table I. Parameter values used in the numerical evaluation

The probability of mission failure $P(C_n)$ is computed using these distributions for the length of stays in failure regions. The other parameters (shown in Table I) are set consistently with the analysis of the fault-tolerant component later in the paper.

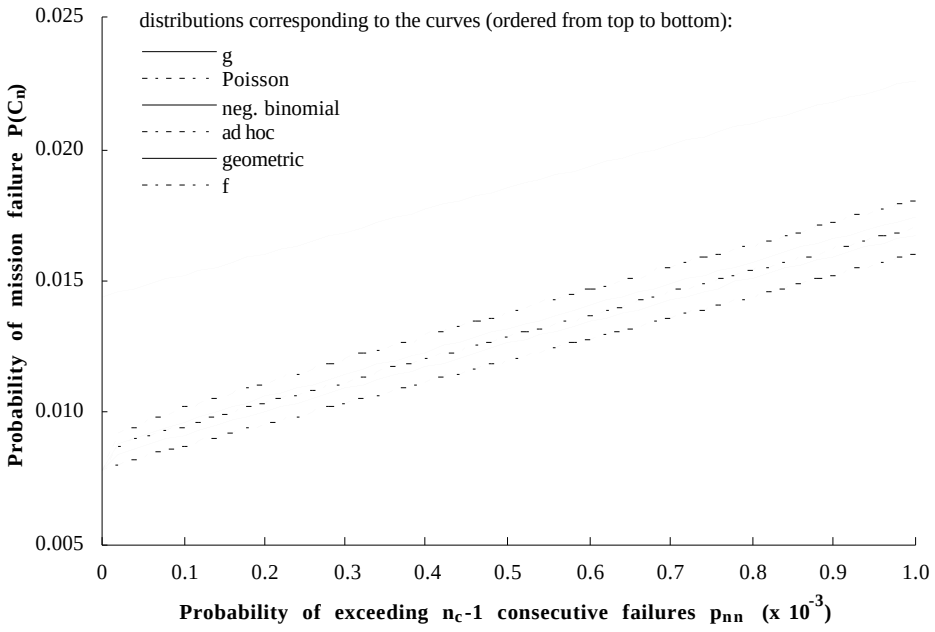

Figure 2. Probability of mission failure as a function of p_{nn}, for different distributions of the duration of failure bursts

Figure 2 shows the probability of mission failure as a function of p_{nn}, in the range $[0, 10^{-3}]$. The non-zero value of $P(C_n)$ for $p_{nn} = 0$ is due to the catastrophic failures. In the case of independence, $p_{nn} = 1.85x10^{-46}$, indistinguishable from 0 in our plots, and $P(C_n) \approx 0.0079$.

Since $p_{bc} > p_{sc}$ has been set, *f* yields better figures than *g*. Increasing n_c would increase this difference. The curves for the four plausible distributions are all enclosed between those of *f*

and *g*; and their distance from *f* depends on the mean duration of stay in the failure regions (and on the difference p_{bc} - p_{sc}). Thus *f* and *g* give bounds on the probability of mission failure, allowing simple tests on the viability of a specific design, given only an estimate of p_{nn}. Last, the value of p_{sb} determines the slope of the curve; increasing p_{sb} increases the probability of entering a failure region and hence the probability of mission failure.

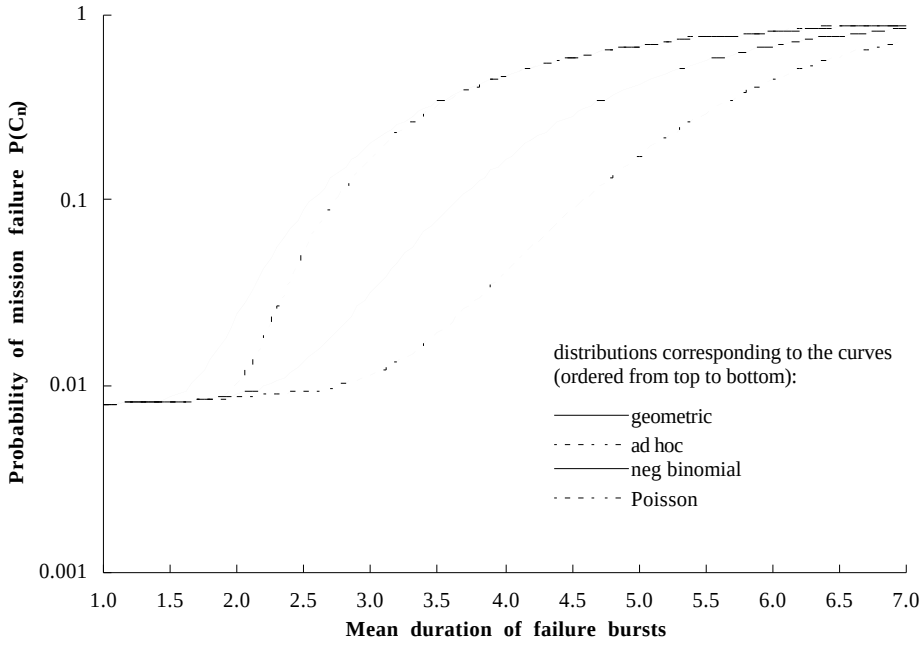

Figure 3. Probability of mission failure as a function of the mean duration of failure bursts

Figure 3 shows the probability of mission failure as a function of the mean stay in a failure region. Observe that, for a given mean, distributions with higher variance cause worse behaviour. The range of parameter values shown extends to unrealistic situations: with such values, the plots show that, to obtain probabilities of mission failure smaller than 10^{-1}, the mean stay in failure regions must be limited to 2.5 -- 4.5 steps, depending on the model.

5 Reliability as function of probabilities for a single iteration. Comparison with the independence case

The results shown thus far derive reliability predictions from the transition probabilities in the Markov models used. Now a more intuitive description is attempted, for comparison with the independence case. The purpose is to show how, "everything else being equal", varying degrees of correlation between the outcomes of successive iterations affect reliability. However, it must be decided what is to "remain equal". The choice made has been to assign the marginal probabilities (per iteration) of success p_s, benign failure p_b and catastrophic failure p_c. These "marginal" probabilities have a clear intuitive meaning: for instance, p_b is the probability

that the given software will suffer a benign failure on one input chosen at random from the operational distribution of inputs. For a given triplet of values for p_s, p_b and p_c, it will be shown how reliability varies, depending on: i) the way failures cluster in time, that is, the correlation among failures over time, and ii) the value of n_c.

Actually, four factors can be identified which affect reliability:

- the locations of the failure points in the input space, i.e., the positions and shapes of the failure regions. This is a characteristic of the software, independent of how it is used (at least if only programs with deterministic behaviour are considered);
- the subdivision of the failure points into those causing benign failures and those causing catastrophic failures. This depends on the characteristics of the controlled system;
- the "input profile" [Musa, et al., 1994], i.e. the probabilities of the different possible input trajectories during operation, which determine the timing and the duration of failure bursts. This probability distribution is certainly a function of the "usage environment" of the controlled system, e.g. for a ship's auto pilot it will be a function of which routes the ships sails, how often and with which pattern of speeds and headings; for an air conditioning system, it will vary with the climate and exposure of the building where it is installed. Once one has chosen the set of possible installations and usage modes for one's software, the probability distribution of trajectories is known, in theory: it can be simulated in practice by testing the software with a simulator of the controlled system. Clearly, the evolution of each individual trajectory is affected by the outputs of the control software itself. While the software behaves correctly, these control outputs can reasonably be considered a function of its specification only. When the software fails, it is the details of the faults that determine which control outputs it produces;
- the robustness of the controlled system, which decides how long a burst of benign failures it can tolerate, i.e., the n_c parameter.

To separate the effects of these factors, an initial model of the behaviour of the program during an infinite succession of missions is given by the Markov chain in Figure 4. This model, like that in Figure 1, allows one to represent independence between successive iterations by setting $p_1 = 1$ and $p_i = 0$ for all $i>1$, but differs from that model in two ways:

1. repeated benign failures cannot cause a catastrophic failure (only "pointwise" catastrophic failures are possible). This model thus describes the effects of the program code and of its usage environment, but not of the robustness of the controlled system, described by the parameter n_c;
2. state C is not an absorbing state: there are transitions from state C, all with the same probabilities as the transitions from S to the same destination states. This represents the

fact that a catastrophic failure terminates a mission, and the next mission starts as after a success. A mission which terminates in state S is followed by another which starts in state S, so this transition is indistinguishable from any other transition from S to S. As the marginal probabilities, p_s, p_b and p_c, can be seen as the average fractions of iterations spent in the states S, B=$\{B_i\}$, or C over infinitely many missions, they are the steady-state probabilities of the different states in this model with approximations that introduce negligible errors [Bondavalli, et al., 1994].

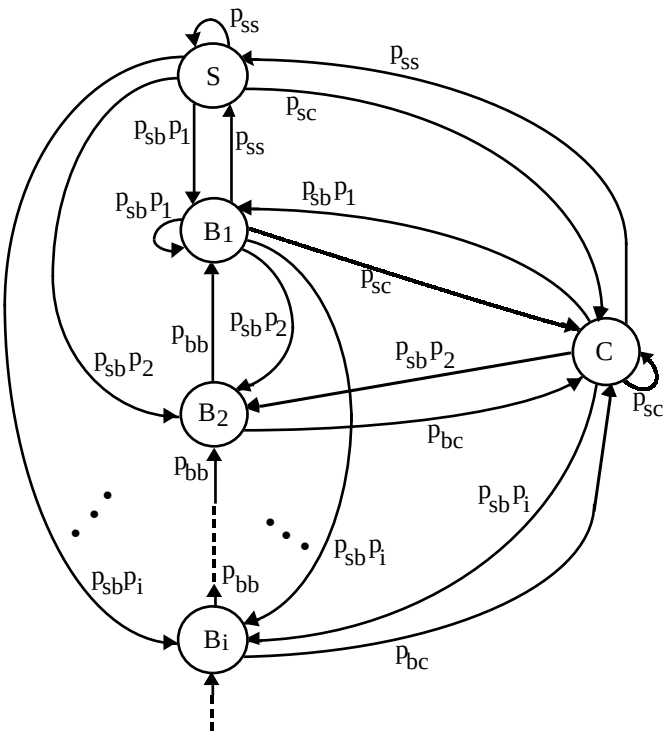

Figure 4. Model of the behaviour of the system during an infinite succession of missions

Notice that to derive these marginal probabilities via testing one could test the software along a realistic distribution of trajectories (e.g. using a simulator of the controlled system), interrupting the execution at each catastrophic failure. However, with that kind of testing, one would be better advised to attempt and measure the reliability directly, rather than the parameters of this (unavoidably unrealistic) model. The purpose of the discussion that follows is to show the interplay of the different probabilistic parameters characterising the behaviour of iterative software.

The standard method for the steady-state solution of the Markov chain [Trivedi, 1982] leads, with the transition rates indicated in Figure 4, to this system of linear equations:

$$\begin{cases} (p_s + p_{b1} + p_c)p_{ss} = p_s \\ (p_s + p_{b1} + p_c)p_{sb}p_i + p_{bi+1}(1 - p_{bc}) = p_{bi}, i = 1,2,3,... \\ (p_s + p_{b1} + p_c)p_{sc} + (p_b - p_{b1})p_{bc} = p_c \\ p_s + p_b + p_c = 1, \quad p_b = \sum_{i=1}^{\infty} p_{bi} \end{cases}$$

In these equations, only the marginal probabilities p_s, p_b, p_c, are known and, obviously, this is not sufficient to determine the transition probabilities. The latter describe the correlation between iterations, and are necessary for computing reliability. In other words, the knowledge that the system tends to be in a certain state a certain fraction of the time does not determine how often it moves into and out of that state. "Correlation over time", that is, "the way failures group into bursts", cannot be fully described by one number.

To visualise the effects of correlation, one can make special assumptions under which it *is* described by one number. In this section, it is assumed that:

i) $p_{bc} = k\, p_{sc}$, with $k>1$ on the basis of the discussion in Section 3.4;

ii) the duration of a failure burst has a geometric distribution with parameter q, as defined in Section 4: all the probabilities p_i can be derived as functions of q. In this model, q=0 represents independence, while q approaching 1 represents the maximum positive correlation, i.e. the case in which a trajectory, after entering a failure region, will remain in it forever.

$\mathbf{p_{ss}}$	$\frac{p_s}{1-p_b q}$
$\mathbf{p_{sb}}$	$\frac{p_b((1-q)(1-p_b q)+kq((1-q)p_b+p_c))}{(1-p_b q)(1+(k-1)p_b q)}$
$\mathbf{p_{sc}}$	$\frac{p_c}{1+(k-1)p_b q}$
$\mathbf{p_{bc}}$	$\frac{kp_c}{1+(k-1)p_b q}$
$\mathbf{p_{bb}}$	$1-\frac{kp_c}{1+(k-1)p_b q}$

Table II. Expressions for the transition probabilities of the model in Figure 4

Under these assumptions, the system above has been solved obtaining all the transition probabilities as functions of the steady state probabilities, the parameter k and the parameter of the geometric distribution, q. These expressions are listed in Table II. These transition probabilities are applied to the model of Figure 4, which is solved for missions that last n executions unless they are interrupted. One can also take account of the fact that sequences of n_c or more benign failures cause a mission to fail, by plugging these transition probabilities back into the model of Figure 1 (the parameter p_{nn} in Figure 1 must be set to $1 - (p_1+p_2+...+p_{n_{c-1}})$) with approximations that introduce negligible errors [Bondavalli, et al., 1994].

Figure 5 shows the curves thus obtained, i.e. the probability of mission failure as a function of the marginal probability p_c, for a geometric distribution of the length of bursts of benign failures, and for different values of the parameter q of the geometric distribution.

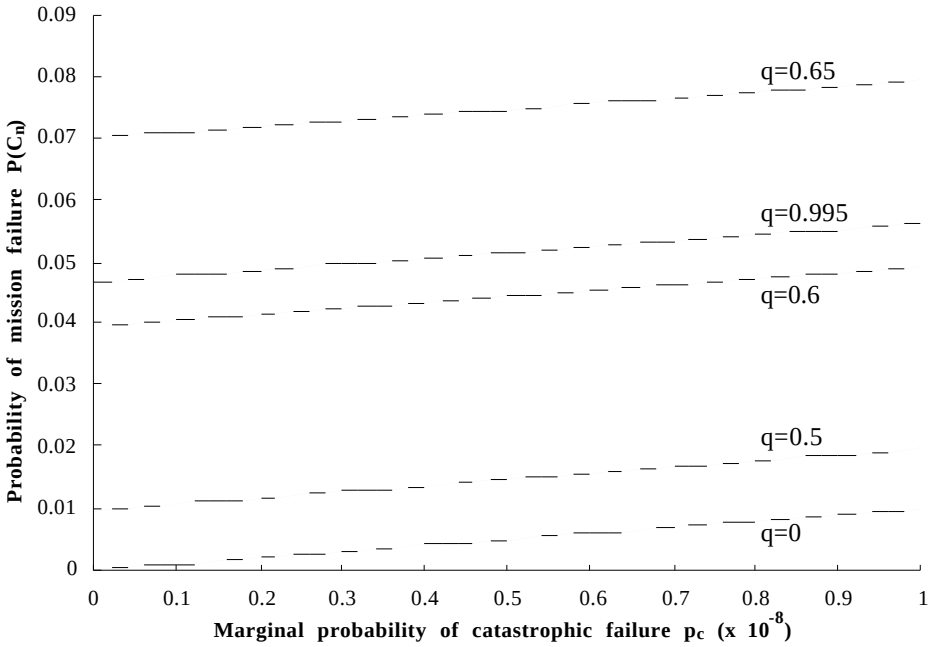

Figure 5. Probability of mission failure, as a function of p_c, for different values of the parameter q of the geometric distribution of the lengths of bursts of benign failures. The other parameter values are $p_s = 10^5 p_b$, i.e. $p_b = (1-p_c)/(10^5+1)$ and $p_s=[(1-p_c)10^5]/(10^5+1)$, k=100 (i.e., $p_{bc}=100\ p_{sc}$) and $n_c=10$.

6 Application to fault-tolerant software

This section introduces an example of "clear box" model of fault-tolerant software, which is then combined with the previous models showing how the reliability predictions change.

6.1 "Clear box" modelling of fault-tolerant software

The evaluation of the reliability of *fault-tolerant* software, specifically of modular-redundant software structures [Avizienis and Kelly, 1984], has been widely pursued by Markov models, as recalled in the Introduction. Many design schemes have been described for software fault tolerance, for example in [Avizienis and Chen, 1977, Laprie, et al., 1990, Randell, 1975, Voges, 1988]. The SCOP scheme [Xu, Bondavalli and Di Giandomenico, 1995] in a configuration resembling a hardware "pair and spare" configuration, as shown in Figure 6, is chosen here as an example.

It employs three variants: two of them are executed in parallel, and their results accepted if in agreement, otherwise the third variant will be executed and voted with the other two.

The modelling approach adopted here was introduced by [Arlat, et al., 1990] using a "reliability submodel" - in the terminology of [Tai, et al., 1996] - to represent the probabilities of the different outcomes at each execution of the redundant component: correct result, detected error, or undetected erroneous result. It was then enriched by Tai and co-authors ([Tai, et al.,

1996] and other papers) with a separate "performance submodel" to describe the timing aspects of the problem, including the probabilities of exceeding the maximum allowed duration of execution.

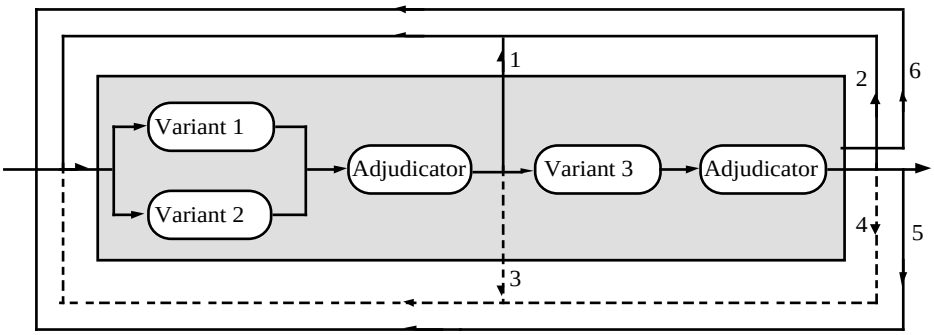

Figure 6. Possible execution paths in SCOP operation. The paths represented in the lower half of the picture imply value failure (undetected for the paths drawn as dashed lines)

Error Types (Events) and Timing parameters	**Probability parameter**	**Value used in the plots**
3 variants err with consistent results	q_{3v}	10^{-10}
2 variants err with consistent results. The third result is inconsistent with them, and may be correct or erroneous.	q_{2v}	$100xq_{iv}^2$
The adjudicator errs and terminates the execution with phase 1, selecting an erroneous, non-majority result	q_{vd1}	$2x10^{-10}$
The adjudicator errs and terminates the execution with phase 2, selecting an erroneous, non-majority result	q_{vd2}	10^{-10}
A variant errs, conditioned on none of the above events happening (i.e., there are one or more detected errors; their statistical independence is assumed)	q_{iv}	variable
The adjudicator errs, at the end of either phase 1 or phase 2, by not recognising a majority (hence causing a detected failure), conditioned on the existence of a majority	q_d	10^{-9}
Parameter of the exponential distributions for the execution times of the variants and adjudicator ($msec^{-1}$)	$\lambda_1 = \lambda_2 = 1/2$, $\lambda_3 = 1/7$, $\lambda_d = 5$	
Period of iterative scheduling (and time-out deadline) for each iteration	τ = 25 msecs	

Table III. Values of the reliability and timing parameters used for the plots of Figures 7-9, where q_{iv} is the independent variable

Details of the model used here for SCOP and its solution method are in [Chiaradonna, et al., 1994]. Table III lists the values of the parameters used to produce the plots of the solutions of

the models (Figures 7-9). This choice of parameter values is, by necessity, arbitrary, though plausible, but it will suffice to show the implications of these models and especially of assuming independence between successive iterations. Choosing $q_{2v} = 100 x q_{iv}^2$ implies essentially that, although the abscissae in the graphs represent the probability q_{iv} of *one* variant failing (at one iteration), the plots are obtained by assuming that the probability of *two* variants failing together with consistent results, causing an undetected failure, is much higher than if the two variants failed independently of each other. So, in the plots, moving right along the horizontal axis represents increasing probabilities of individual failures of the variants, while preserving positive correlation between them.

It is assumed that the variant to be executed only if an error is detected (variant 3 in Figure 6) is the slowest one. The average duration of an iteration and the probability of time-out, determined using the "performance submodel", increase with the probability of errors in the results of a variant, but with minimal effect for the range of situations considered here. The mission duration is 10^6 iterations, a reasonable order of magnitude for, e.g., a workday in non-continuous process factory operation, or a flight for civil avionics (with an execution rate of a few tens of iterations per second).

The probability of mission failure, i.e., of having at least one undetected error in a mission, is then an approximately exponential curve (bottom curve in Figure 7, corresponding to q=0). As shown, this choice of parameter ranges extends to unrealistically low probabilities of success of a mission. The curve crosses the vertical axis at a value of the ordinates greater than 0 (which is not apparent in the plots), representing the failures due to coincident errors of three variants and/or of the adjudication.

6.2 Extension to model failure sequences without independence

The predictions of the "clear box" model for the outcome of an iteration can now be "plugged into " the "black-box" model to take account of the degrees of "burstiness" of the failure process and robustness of the controlled system. The special case of the "black-box" model studied in Section 5, in which the lengths of bursts of benign failures has a geometric distribution, is used.

The SCOP fault-tolerant component is considered, still assuming $q_{2v} = 100 \times q_{iv}^2$ so that q_{iv}, the failure probability per execution of the variants, can be used as a combined measure of "unreliability" across a hypothetical family of programs. Then, p_s, p_b and p_c are obtained as a function of q_{iv} and of the parameters in Table III. The "black-box" transition probabilities of Figure 4 are a function of p_s, p_b, p_c and of the parameter q of the geometric distribution. Setting $n_c = 10$ and k=100, one can derive the transition probabilities as in the previous section and apply them back into the model of Figure 1. The curves of the probability of mission

failure for the SCOP fault-tolerant component are then obtained, and plotted in Figure 7, as a function of q_{iv}. The curves correspond to different levels of correlation between iterations, represented by the parameter q.

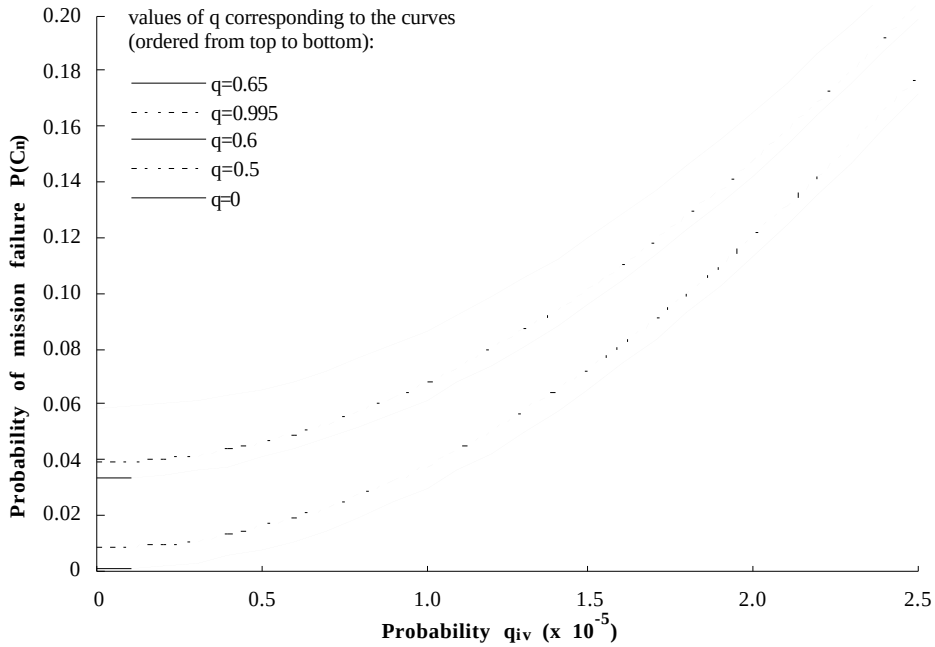

Figure 7. Probability of mission failure for the SCOP component, as a function of q_{iv}, for different values of correlation between iterations, with $q_{2v} = 100 \times q_{iv}^2$

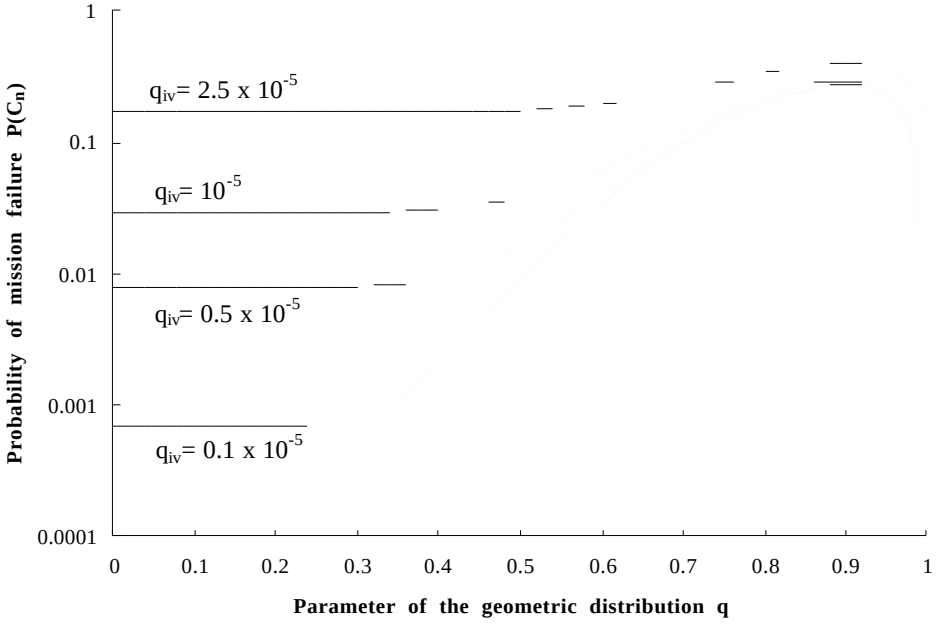

Figure 8. Probability of mission failure for the SCOP component as a function of q for different values of q_{iv}, with $q_{2v} = 100 \times q_{iv}^2$

Figures 7 and 8 show that decreasing values of q, approaching 0, reduce the probability of mission failure, as they represent decreasing positive correlation between successive iterations. They also show a peculiar effect of a strong positive correlation among successive benign failures: when q approaches 1, i.e., the correlation between successive failures tends to 1, the system tends to enter the B state very rarely, and to stay there for a very long time. So, each entry in the B state effectively implies mission failure, but this event happens in a very small fraction of the missions.

As q approaches 1, the simplifying assumptions made may no longer apply and a model with the true probability distributions would then be necessary for precise predictions. Still, referring to Figure 1, a value of q close to 1 implies that each iteration starting from state S may initiate mission failure with probability $(1-p_{ss})= p_{sb}+p_{sc}$ (as the term $p_{nn}p_{sb}$, on the arc from B to C in Figure 1, tends to p_{sb}). If systems with the same probability of *entering* state B, p_{sb}, were compared, a higher q would mean a system with a higher probability of mission failure. Since the systems compared are those with the same probability of *being* in state B, a higher q means a system with a very low p_{sb}.

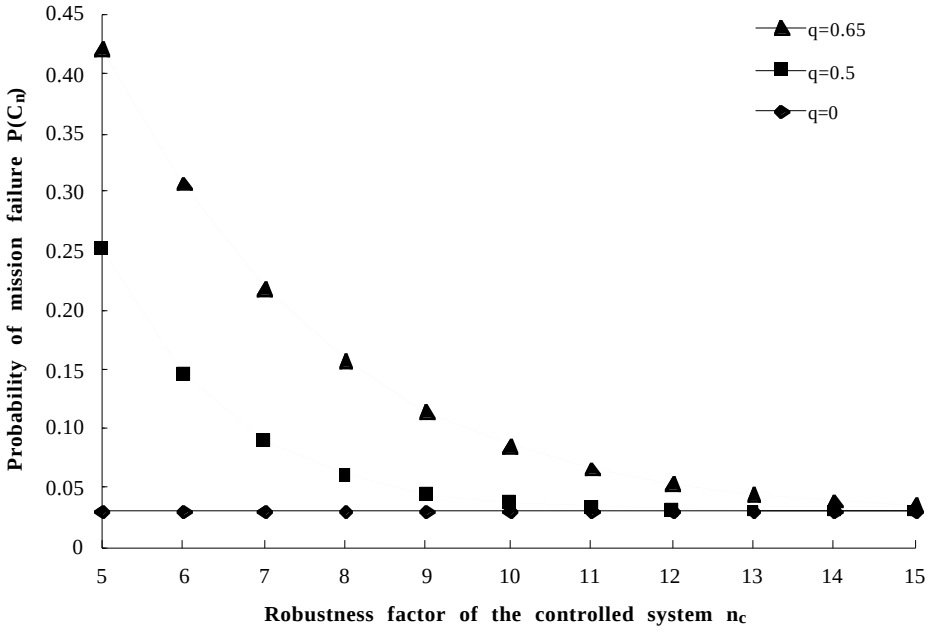

Figure 9. Probability of mission failure for a fault-tolerant component built using SCOP, as a function of n_C, for different values of correlation between iterations

Last, Figure 9 shows how the probability of mission failure varies as a function of the robustness of the controlled system, described by the parameter n_C. Here, the value of q_{iv} is set to 10^{-5} and three different values of q are considered. Given a distribution of the length of failure bursts, increasing n_C reduces p_{nn}, the probability of n_C or more benign failures in a row,

which would cause mission failure. Again, decreasing values of q, approaching 0, reduce the probability of mission failure. In the setting chosen, in the case of independence (q=0), sequences of failures have negligible influence on the probability of mission failure.

7 Conclusions

Evaluating the reliability that can be obtained with alternative design solutions is obviously desirable. In the modelling of software with iterative execution (e.g., process control software), this paper improves on the previous literature by studying the effects of both:

1. the tendency of failures to appear in bursts (longer than independence between successive iterations would imply), and
2. the ability of a controlled system to tolerate short bursts of failures.

Factor 1 has been modelled in terms of assumptions on the characteristics of failure bursts caused by connected "failure regions" in the input space. Including factor 2 causes a more complex interplay among several effects, which have been described singularly and then in combination.

The main contribution of this paper is some added insight in this complex probabilistic problem: which factors tend to increase or decrease the reliability of a system, and how. This added insight clarifies possible pitfalls when "plausible" reliability models are used for prediction, as a substitute for measuring reliability directly. For instance, serious errors may be incurred by depending on the marginal probability of failure while ignoring correlation among successive failures, or considering only some of the effects of long failure bursts.

These indications are also useful for designers who believe that they can control some of the factor described. For instance, if one does not know how to further improve the probability of failure per execution, it may be worthwhile to spend some effort in avoiding (through error detection and recovery) long bursts of "benign" failures. It may be convenient for a designer to accept a higher probability of failure per execution in return for a lower probability of long bursts of failures; or to pursue the latter so as to be able to build less resilience (a lower n_C) into the controlled system.

As for producing actual reliability predictions, these models, like all previous "structural" models of software reliability, pose difficult parameter estimation problems. To describe important phenomena that previous authors had ignored, additional parameters have been introduced. Solving the models shows that these factors make considerable difference for the reliability of a system, i.e., that the additional parameters are indeed necessary for proper prediction. Estimating them is possible in theory, but very difficult. On the one hand, these parameters describe distributions of the characteristics of software faults - the relative

frequencies of benign *vs* catastrophic failures, the shapes and sizes of failure regions, etc. - that can all be observed and measured, after a software fault has been discovered (cf. [Amman and Knight, 1988, Bishop, 1993]). By devoting resources to analysis and measurement, sufficient statistical knowledge could eventually be built about these distributions. On the other hand, this process might be prohibitively long and, in any case, such measurement campaigns are rarely attempted even in high-quality development organisations.

Areas where precise estimation is less necessary have also been highlighted: for instance, the probability distribution functions *f* and *g* for the duration of failure bursts (Section 4) give upper and lower bounds on the reliability obtainable with a given probability of non-catastrophic failure bursts.

As usual, some caution is required about modelling assumptions. Those chosen here are based on plausible considerations, but they should not be used as though they were empirical knowledge about any set of real programs. In particular, the restrictive assumptions in Sections 5 and 6 do not describe the general behaviour of software. By eliminating some degrees of freedom in the behaviour of a system, they allow the results of modelling to be visualised more easily; but more degrees of freedom still exist in reality. A designer wishing to estimate bounds on the behaviour of a specific system should consider the effects of varying all parameters without arbitrary restrictions.

An interesting special case for the application of these models would be that of a program for which a testing strategy can be devised to estimate an upper bound on (or the distribution of) the sizes of failure regions and hence, given some knowledge about the dynamics of the controlled system, on the duration of failure bursts. The "black box" models discussed apply to all software with iterative executions. In addition, Section 6 contains a novel contribution to the evaluation of fault-tolerant software, for which Markov reliability models have been widely published.

References

Amman, P. E. and Knight, J. C., (1988)
"Data Diversity: An Approach to Software Fault Tolerance,"
IEEE Transactions on Computer, **C-37** (4), pp. 418-425.

Arlat, J., Kanoun, K. and Laprie, J. C., (1990)
"Dependability Modelling and Evaluation of Software Fault-Tolerant Systems,"
IEEE Transactions on Computers, **C-39** (4), pp. 504-512.

Avizienis, A. and Chen, L., (1977)
"On the Implementation of N-Version Programming for Software Fault Tolerance During Program Execution,"
in Proc. 1st Int. Conf. on Computing Software Applications (COMPSAC-77), pp. 149-155.

Avizienis, A. and Kelly, J. P. J., (1984)
"Fault Tolerance by Design Diversity: Concepts and Experiments,"
IEEE Computer, **17** (8), pp. 67-80.

Bishop, P. G., (1993)
"The Variation of Software Survival Time for Different Operational Input Profiles (or why you Can Wait a long Time for a big Bug to Fail),"
in Proc. 23th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-23), Toulouse, France, pp. 98-107.

Bishop, P. G. and Pullen, F. D., (1988)
"PODS Revisited - A Study of Software Failure Behaviour,"
in Proc. 18th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-18), Tokyo, Japan, pp. 1-8.

Bondavalli, A., Chiaradonna, S., Di Giandomenico, F. and Strigini, L., (1994)
"Modelling Correlation Among Successive Inputs in Software Dependability Analyses,"
CNUCE-CNR Technical Report N. C94-20, (Obtainable from first author and URL: http://bonda.cnuce.cnr.it/Documentation/Reports/Reports.html), 1994.

Bondavalli, A., Chiaradonna, S., Di Giandomenico, F. and Strigini, L., (1995)
"Dependability Models for Iterative Software Considering Correlation among Successive Inputs,"
in Proc. IEEE Int. Computer Performance and Dependability Symposium (IPDS'95), Erlangen, Germany, IEEE Computer Society, pp. 13-21.

Brilliant, S. S., Knight, J. C. and Leveson, N. G., (1990)
"Analysis of Faults in an N-Version Software Experiment,"
IEEE Transactions on Software Engineering, **SE-16** (2), pp. 238-247.

Chen, S. and Mills, S., (1996)
"A Binary Markov Process Model for Random Testing,"
IEEE Transactions on Software Engineering, **SE-22** (3), pp. 218-223.

Chiaradonna, S., Bondavalli, A. and Strigini, L., (1994)
"On Performability Modeling and Evaluation of Software Fault Tolerance Structures,"
in Proc. 1st European Dependable Computing Conference (EDCC-1), Berlin, Germany, Springer-Verlag, pp. 97-114.

Csenki, A., (1991)
"Recovery Block Reliability Analysis with Failure Clustering,"
in Proc. 1st IFIP Int. Working Conf. on Dependable Computing for Critical Applications (DCCA-1), Santa Barbara, California, 23-25 August 1989, Dependable Computing and Fault-Tolerant Systems, Vol. 4 - Springer Verlag, pp. 75-103.

IEEE-TR, (1993)
"Special Issue on Fault-Tolerant Software,"
IEEE Transactions on Reliability, **R-42** (2), pp. 177-258.

Knight, J. C. and Leveson, N. G., (1986)
"An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming,"
IEEE Transactions on Software Engineering, **SE-12** (1), pp. 96-109.

Laprie, J. C., Arlat, J., Beounes, C. and Kanoun, K., (1990)
"Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures,"
IEEE Computer, **23** (7), pp. 39-51.

Littlewood, B. and Miller, D. R., (1989)
"Conceptual Modelling of Coincident Failures in Multiversion Software,"
IEEE Transactions on Software Engineering, **SE-15** (12), pp. 1596-1614.

Musa, J. , Juhlin, B. , Fuoco, G. , Kropfl, D. and Irving, N., (1994)
"The Operational Profile,"
in "Handbook of Software Reliability Engineering", M. R. Lyu Ed., McGraw-Hill and IEEE Computer Society Press, pp. 167-216.

Randell, B., (1975)
"System Structure for Software Fault Tolerance,"
IEEE Transactions on Software Engineering, **SE-1** (2), pp. 220-232.

Strigini, L., (1996)
"On Testing Process Control Software for Reliability Assessment: the Effects of Correlation Between Successive Inputs,"
Software Testing Verification and Reliability, **6** (1), pp. 36-48.

Tai, A. T., Meyer, J. F. and Avizienis, A., (1996)
"Software Performability: from Concepts to Applications,"
Kluwer Academic Publishers, USA.

Tomek, L. A., Muppala, J. K. and Trivedi, K. S., (1993)
"Modeling Correlation in Software Recovery Blocks,"
IEEE Transactions on Software Engineering, **SE-19** (11), pp. 1071-1085.

Trivedi, K.S., (1982)
"Probability & Statistics with Reliability, Queuing, and Computer Science Applications,"
London, Prentice-Hall.

Voges, U., (1988)
"Software Diversity in Computerized Control Systems,"
Dependable Computing and Fault-Tolerance Series, A. Avizienis, H. Kopetz and J. C. Laprie Ed., Vol. 2, Wien, Springer-Verlag.

Xu, J., Bondavalli, A. and Di Giandomenico, F., (1995)
"Dynamic Adjustment of Dependability and Efficiency in Fault-Tolerant Software,"
in "Predictably Dependable Computing Systems", B. Randell, J. C. Laprie, H. Kopetz and B. Littlewood Ed., Springer-Verlag, pp. 155-172.