Modelling the Effects of Input Correlation in Iterative Software

A. Bondavalli[1], S. Chiaradonna[2], F. Di Giandomenico[2], S. La Torre[3]

[1] **CNUCE-CNR, Italy** [2] **IEI-CNR, Italy** [3] **University of Salerno, Italy**

Address for correspondence and proofs:

Felicita Di Giandomenico, IEI-CNR, Via S. Maria 46, 56126 Pisa, Italy.

Abstract

This paper deals with the dependability evaluation of software programs of iterative nature. In this work we define a model that is able to account for both dependencies between input values of successive iterations and the effects of sequences of consecutive software failures on the reliability of the controlled system. Differently from previously proposed models, some effort is devoted to address the problem of how to get accurate estimates for the basic parameters. A model is thus proposed that, requiring the designers or users to provide information usually obtainable by experimental techniques, e.g. testing, is more useful and more generally applicable. Then a thorough analysis is performed to highlight the effects of the different parameters on the dependability attributes. This analysis allows to appreciate which effects (and their extent) have variations of both correlation between successive inputs and different structural characteristics of the software at hand. Moreover the robustness of the model against imprecise assessments of the starting parameters is also shown.

1 Introduction

This paper deals with the dependability modelling and evaluation of software programs of iterative nature, typically software which controls the activity of some (physical) system. Most control software has the following basic structure: the software performs a repetitive cycle of sampling sensors and control input, using several control laws to compute the proper response to the different actuators and then sending appropriate commands to the actuators. The focus is here in modelling the behaviour of such software and analysing it over a specific time interval, which constitutes a system mission.

The effective utility of a model depends on many factors, that may also relate to its intended use. These include the realism, i.e. the plausibility of the assumptions made, but also very important are the ability to account for the relevant basic details, the robustness against inaccurate values assigned to some parameters and the possibility or easiness to obtain proper estimation of the parameters. A set of plausible values for the model parameters is normally derived from testing or from previous experience with similar software. Usually, models kept simple by numerous assumptions, have a limited realism although they may use parameters with a more intuitive meaning and in some cases easier to determine than those required by more realistic models.

The dependability of programs of iterative nature (as well as that of other software structures) is usually analysed using models that assume independence between the outcomes of successive executions of a program [1, 2, 3, 4]. This assumption, which is often false for many applications, strongly limits the realism of these models although it makes the associated mathematics simpler [4]. In fact, it allows to use constant probabilities of failure and success at each iteration for the entire mission duration, with the additional advantage that such probabilities are easily estimable, e.g. through testing. However, experiments and theoretical justifications show the existence of contiguous failure regions in the program input space and that, for many applications, such as real-time control systems where iterations have usually short cycle duration, the inputs often follow a trajectory of close points in the input space. For these reasons the inputs which originate failures of the software are very rarely isolated events but more likely grouped in *clusters* [5, 6, 7]. For all the classes of applications to which these considerations apply, analyses of software dependability performed assuming independence between successive iterations seem to lead to results excessively diverging from the real behaviour of the analysed system. Another important characteristic of these applications that should be captured by a realistic model (and usually is not) is the effect of clustering of failures of the software on the system mission. The effects of software failures on the (physical) system are usually considered one by one: a software failure either determines the failure of the system mission (commonly indicated as catastrophic failure) or brings the system in a temporary undesirable, but still recoverable, condition (benign failure). However, many systems, although able to tolerate isolated or short bursts of benign failures, may not be able to survive to the lack of feed-back control for too long. Hence, the occurrence of long sequences of even benign failures often cause actual damage on the system (from stopping a continuous production process to letting an airplane drift out of its safe flight envelope).

In this paper we offer two contributions to the modelling of both dependencies between input values of successive iterations and the possibility that repeated, non fatal, failures may together cause mission failure. The first concerns the problem of providing accurate estimates for the basic parameters of the model. Instead of restricting to the definition of a model with the only purpose of understanding the underlying mechanisms ruling the behaviour of a system in which successive inputs are correlated (as was done in [8]), we also consider the usability of the model itself. We thus develop a new modelling framework using as basic knowledge information that appears to be relatively easy and cheap to determine by experimental techniques if compared with the difficulty to assess the system dependability indicators of interest. In this way, an interesting compromise is reached between a fairly realistic model for obtaining predictions of dependability attributes of a system and difficulty (and costs) in obtaining the basic knowledge necessary to resolve the model. The second contribution is an extensive analysis of the developed model. The evaluation performed here allows to appreciate the kind and the extent of the effects of the various parameters on the reliability of iterative software. We analysed the sensitivity of the model to three types of parameters representing:

i) the correlation between successive inputs,

ii) the different structural characteristics of the software at hand, and

iii) our starting parameters in order to check the robustness of the model against inaccurate initial assessments.

The rest of the paper is organised as follows. Section 2 contains a brief recall of the literature and a discussion on the problem of how values for model parameters may be obtained. In Section 3, the assumptions and the dependability measures we are interested in are first described, and then a general model for iterative software is developed together with our proposed approach for obtaining accurate estimations of the parameters. Evaluations for specific classes of iterative software are presented in Section 4. They show the effects of varying the parameters expressing the correlation between successive inputs and the robustness of the model against imprecise assessments of the starting parameters. Our evaluations include also the case of independence among successive inputs when relevant for comparison purposes. Finally, Section 5 summarises our conclusions.

2 Background

The problem of modelling and evaluating the effects of correlation between the outcomes of successive iterations has been addressed in the literature [8, 9, 10]. Csenski [9] models the behaviour of a recovery block structure, first defined in [11] and subject of many papers afterwards, composed of a primary version, an alternate version and a perfect acceptance test. Failures of the primary module are distinguished in :

i) point failure: when the input sequence enters a failure region,

ii) serial failure: a number of consecutive failures occurring after a point failure, i.e., after that the input trajectory enters a failure region.

The number of serial failures subsequent to any point failure is a random variable. From these modelling assumptions, a simple Markov chain with discrete time is developed allowing an analytical evaluation of the reliability (MTTF) of the recovery blocks. Tomek and co-authors [10] analyse the different forms of correlation of the recovery blocks structure, including correlation among the different alternates and among alternates and the acceptance test on the same inputs. While the previous two papers limit their modelling to the correlation between inputs, our work [8] considers also the possibility that repeated, non fatal software failures may together cause the failure of the system mission.

However, the motivation in [8] was to understand the underlying mechanisms ruling a model in which correlation between successive inputs is considered, rather than the definition of models really usable. The starting parameters for that model are, in fact, the state transition probabilities of the software system in its operational profile, whose estimation through experimental techniques, e.g. testing or fault injection, is undoubtedly very difficult to obtain if possible at all, and most of the times useless since the effort required may be comparable to estimate the dependability figures of interest. These remarks motivate our interest in developing an alternative process to obtain the same final evaluations as in [8], but relying on information which can be derived by experimental methods in a simpler and cheaper way.

Experimental techniques have been proposed recently as an alternative approach to model-based evaluation for assessing the dependability of critical systems. Traditional usage of testing and fault injection has been aimed at improving the dependability of a system: testing allows to identify residual faults to be corrected, while fault injection reveals faults which are not properly tolerated by the system. Recently, both techniques have been also proposed as means for evaluating system dependability: testing by revealing residual faults [12, 13] and fault injection to evaluate coverage and latency with respect to an assumed set of faults which can affect the system and its components [14, 15]. However, using these techniques for evaluation purposes turns out to be significantly difficult and expensive. First, one has to collect enough experimental results so that accurate statistical inference about the dependability figures of interest can be drawn. Second, the effectiveness of these methods involves issues such as choosing the most appropriate inference procedure or deriving a realistic approximation of the user operational profile. The knowledge of the characteristics of the system and the program to experiment upon is, in fact, crucial. In the extreme case, once all the details about the operational environment are known, one could derive directly, by using experimental techniques, estimates of the final quantities of interest. But, also in this ideal extreme case, the high costs involved make the derivation of the final estimates using experimental techniques prohibitive.

An hybrid approach, where experimentation and analytical models are combined, seems to be very effective, as addressed in [16], where the interactions between analytical dependability modelling and experimental evaluation are discussed. Following this approach, at first experimental evaluation (e.g., testing) is applied to derive basic information about the software under analysis (thus making the implied costs affordable), which is then used as starting parameters of an analytical model whose solution gives the dependability figures desired. This approach is common sense for systems designed according to some well understood and proved-to-be-correct structure, where, instead of considering the system as a black-box, its internal structure is taken into account as well. For example, a wide variety of models exist in the literature, such as [14, 17], to derive the dependability figures of fault tolerant structures starting from the figures of their components. In such cases, testing (or fault injection) can be used to obtain an estimate of the probability of failure/success of the individual components of the software system.

3 The Model

3.1 Assumptions and Dependability Measures

Software (seen as a black box) of an iterative nature is assumed, as typically used in control systems. The system is controlled by a static periodic scheduling policy such that each iteration is started cyclically after a time interval of constant width τ and is aborted by a watch-dog timer should it last more than the time threshold τ. Thus, a *mission,* of a given duration T, may be considered as composed of a constant number $n = T/\tau$ of iterations.

At each iteration, the program accepts an input and produces an output. The outcomes of an individual iteration may be: i) *success*, i.e., the delivery of a correct result, ii) a *benign failure* of the program, i.e., an output that is not correct but does not, by itself, cause the entire mission of the controlled system to fail, or iii) a *catastrophic failure*, i.e., an output that causes the immediate failure of the entire mission. From the software viewpoint solely, and without referring to any specific application, we assume here that all detected failures (default safe values of the control outputs from the computer) do not prevent the mission to continue and are in this sense benign, whereas undetected failures are conservatively assumed to have a "catastrophic" effect on the controlled system. Obviously, if knowledge of the consequences of software failures on the system was available for a specific system, the proper splitting of software failures into benign and catastrophic could be precisely made.

Failure regions, consisting of contiguous points, are subsets of the program input space for which the output produced violates the specifications. In 7, besides showing that for some programs failure regions are made of contiguous points and providing some theoretical justifications for this conjecture, it is also shown that the shapes can be often angular, elongated and rectangular.

The successive inputs form a "trajectory": any input value is assumed to be close (but not necessarily contiguous) to the previous one. We have a so called random or deterministic walk trajectory with a small step length. The step length, i.e., the distance between two successive input points, is considered small if the difference of the values of the two points on each dimension of the input space is small compared to the size of the input space in that dimension. If the step length becomes comparable to the size -in each dimension- of the input space (e.g., 50%) then, as shown in 6, we obtain uniform distribution of the inputs and therefore independence. In such a context many different 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 random small quantity, 2) (subcase of 1) a "forward-biased" trajectory: passing from one input to the next the direction may only change slightly, 3) (subcase of 2) a trajectory of points on a straight line, at a random, small distance from each other.

The hypotheses we make in modelling sequences of correlated failures are:

1) sequences of benign failures equal or longer than a threshold n_c, $n_c > 0$, cause the failure of the entire mission, thus having the same effect as a catastrophic failure. A single success before the n_c-th consecutive failure will bring the system into a stable state, i.e. the memory

of the previous failure sequence is immediately lost. Of course, the assumption that any sequence of up to n_c-1 failures will be tolerated, and all longer sequences will be catastrophic, is still a simplification of reality, yet more realistic than assuming that a controlled system can tolerate any arbitrary series of benign failures. The purpose of assuming that a single success before the n_c-th consecutive failure causes the loss of the memory of the previous failures is to keep the developed model easy to understand. In any case there are no particular difficulties to extend the model to represent the need of a given number of successes to loose memory of the previous failures;

2) the trajectory of the input sequence is "forward-biased": passing from one input to the next the direction may vary with a small angle. Actually, many applications (e.g., radar systems or navigation systems) control systems possessing physical inertia. In these cases, the successive values of many physical quantities provided by sensors and used as inputs by the control software show "forward-biased" trajectories;

3) the failure regions are convex and separated by each other: to pass from one to another the input trajectories must cross at least one point for which the program executes successfully. This separation of failure regions reflects the users view: they perceive a set of contiguous failure points as one single region. A designer or analyst would have a different perspective: they would characterise a failure region as the set of inputs for which the same fault of the software is activated. In this case, failure regions need not to be made of contiguous points, regions may overlap and may not surrounded by points for which the program executes successfully. Assuming convex failure regions determines that our forward biased trajectories, once they have left a failure region, are unlikely to re-enter soon the same failure region just left. We can thus consider in our models the probability of entering a failure region as a constant. This is actually the main rationale for choosing convex regions: since there is no evidence for choosing particular shapes on a general basis, our choice is driven by the necessity to simplify the modelling. It is however clear that our model represents an approximation of the real system behaviour when concave failure regions are considered, since in this case the probability of entering a failure region varies depending on the trajec-

tory being close to the border of a concave failure region. Moreover the assumption of convex failure regions is also conservative in the sense that, for a given size of a failure region, trajectories become more likely to stay longer in the region.

Among the various attributes of dependability, we restrict here on the probability of surviving a mission (which in our particular case of static periodic scheduling corresponds to the reliability after a certain number of executions). Other dependability attributes, like performability and safety, are not so interesting in our scenario. In our context a mission is composed of a constant number of iterations and only two different accomplishment levels, besides mission failure, are defined, thus the performability measure 18 purely reflects the reliability. To verify this, in 19 the expression of the performability was derived, based on a simple reward function, and its numerical evaluation performed. As expected, the results indicated the same trends for the reliability and the performability, thus showing the uselessness of performing a performability analysis. On the other side, an effective estimation of the safety of computer controlled systems would require a wide knowledge of the system itself and not just of its software. It is very difficult to determine which software failures have catastrophic consequences on the entire system. Moreover this analysis must be performed on the specific application considered and cannot be done on a general basis.

3.2 Derivation of Estimates for the Basic Parameters

Now we describe the procedure to follow for providing estimates for the basic parameters of the model for the software under analysis that will be presented in the next subsection.

We assume that a realistic distribution of inputs can be generated by synthesising the population of trajectories expected for our system in its operational profile. Individual inputs are then used to test the program in a testing regime that is an approximation of the operational environment of the system under analysis. In detail, differently from the operational environment, here: i) the effects of sequences of benign failures are not taken into account (no mission failures are due to sequences of benign failures); and ii) missions are not of a fixed duration but they are terminated only by the occurrence of a pointwise catastrophic failure (after a catastrophic failure the program is reset to the initial state). A sufficiently long test activity on individual inputs, where many missions are linked together, allows to determine quite easily the probabilities p_s, p_b and p_c of success, benign failure and pointwise catastrophic failure respectively, which can be interpreted as "steady state probabilities" relative to the input distribution. Aiming at the determination of steady state probabilities does not require accounting for the specific paths leading to the selected state. The estimation of probabilities on the specific paths is instead the information that must be collected to solve models requiring values for the state transition probabilities. Therefore, having to collect less information and using individual inputs rather than trajectories, makes testing for determining steady state probabilities much cheaper than for determining state transition probabilities.

The steady state probabilities of the system in this testing regime, together with parameters expressing the correlation and structural properties, are used to find an accurate estimate of the state transition probabilities of the software (which are the same both for the testing regime and the operational environment). Clearly, the quality of the measured probabilities heavily depends on how properly the inputs used for testing represent trajectories encountered by the program during its operational lifetime. It must be also noticed that here we assume that the steady state probabilities p_s, p_b and p_c have been determined by testing although they could be provided following alternative approaches. Actually, from the point of view of our modelling effort, only the fact that this knowledge can be reasonably provided really matters.

The model described by the Markov chain in Figure 1 represents the system in the testing regime and shows more details than strictly implied by the testing activity. The state representing the benign failure is split in an infinite number of states B_i, where the state B_i is reached from S when the input trajectory enters a failure region and remains in it for i iterations (if no catastrophic failures are encountered). The state transition probability from state S to state B_i is defined as $p_{sb} \cdot p_i$, where p_{sb} represents the probability to enter a failure region and p_i the probability to encounter a failure region of length i. Then, the steady state probability of benign failure must be seen as the sum of the probabilities p_{bi} of being in state B_i, that is $p_b = \sum_{i=1}^{\infty} p_{b_i}$.

Transitions from state C to states S, B_i and C, all with the same probabilities as transitions from S, model the reset of the software system to the initial state after the occurrence of a catastrophic failure. We account for the assumption that each trajectory, after crossing a failure region, experiences at least one successful execution: B_1 is the state reached after the last failure in the crossed region and we assign the value 1 to the arc from the state B_1 to S. State S represents a success and when it is reached the system looses memory of previous possible benign failures. If instead of needing a single success to cancel the effects of the previous failures, N_S successes would have been required, then the state S would have been substituted by a set of N_S states S_i, connected with arcs to both states B_i and the state C. The only complication to solve the resulting model would be the mathematics, but no particular problems would arise.

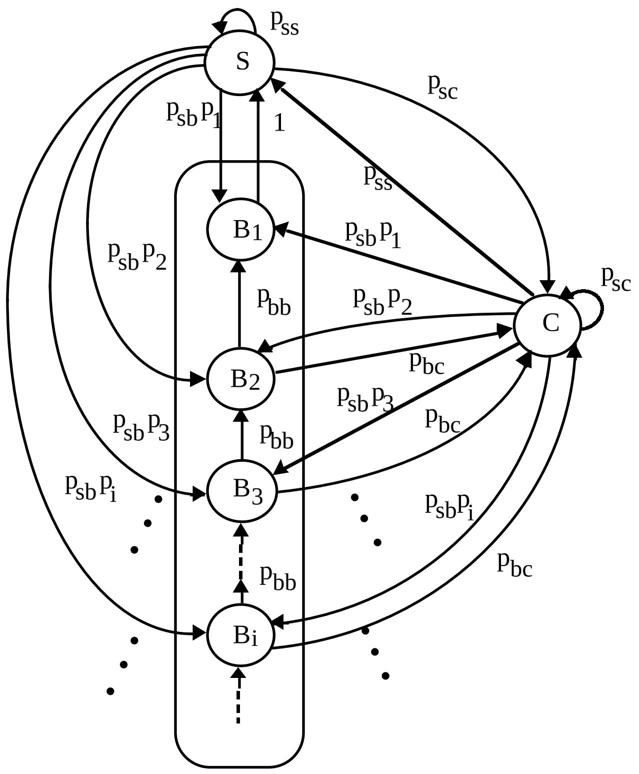

Figure 1. **The model used to derive the state transition probabilities.**

This general model takes into account the correlation between successive inputs through the parameters p_i on the arcs from S to the states B_i, without being tied to any specific distribution representing the permanence of the input trajectory in failure regions. The following matrix P is its transition probability matrix.

$$
P=\begin{array}{c|cccccccc|}
 & S & B_1 & B_2 & \cdots & B_i & B_{i+1} & \cdots & C \\
S & p_{ss} & p_{sb}\cdot p_1 & p_{sb}\cdot p_2 & \cdots & p_{sb}\cdot p_i & p_{sb}\cdot p_{i+1} & \cdots & p_{sc} \\
B_1 & 1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\
B_2 & 0 & p_{bb} & 0 & \cdots & 0 & 0 & \cdots & p_{bc} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & p_{bc} \\
B_i & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & p_{bc} \\
B_{i+1} & 0 & 0 & 0 & \cdots & p_{bb} & 0 & \cdots & p_{bc} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
C & p_{ss} & p_{sb}\cdot p_1 & p_{sb}\cdot p_2 & \cdots & p_{sb}\cdot p_i & p_{sb}\cdot p_{i+1} & \cdots & p_{sc}
\end{array}
$$

For all the distributions p, such that the Markov chain is irreducible, aperiodic and with all recurrent non-null states, the vector (p_s, p_{b1},....,p_{bi},...., p_c), representing the steady-state distribution of the probabilities of staying respectively in the states S, B_1,......., B_i,......, and C, is the unique solution to the equation system 3.1:

$$
\begin{cases}
(p_s,p_{b_1},\dots,p_{b_i},\dots,p_c)=(p_s,p_{b_1},\dots,p_{b_i},\dots,p_c)\cdot P \\
p_s+\sum_i p_{b_i}+p_c=1
\end{cases}
\qquad 3.1
$$

Our aim is now to derive the transition probabilities (p_{ss}, p_{sc}, p_{sb}, p_{bb} and p_{bc}), but, unfortunately, the available information, that is the probabilities p_b, p_s and p_c plus the p_i's representing the distribution of the length of staying in failure regions, is not sufficient. Further information may be obtained by analysing the system; for example such analysis could suggest special relationships between transition probabilities. Some reasonable examples are the following:

a) a control software for which catastrophic failures occur independently from the trajectory being crossing failure regions, thus the probability of a catastrophic failure is the same if the last execution produced a benign failure or a success. Setting $p_{bc} = p_{sc}$ in our model allows to represent this case;

b) a control software for which the probability of the next iteration producing a catastrophic failure increases if the last iteration produced a benign failure. This is modelled by setting $p_{bc}>p_{sc}$. This looks like a realistic assumption in many cases: for instance, one may assume that a benign failure implies that the program has entered a region of its input space where failure is especially likely, and that a fixed proportion of such failures happens to be immediately catastrophic, or a system suffering from imperfect recovery such that when the recovery procedure is activated, failures are more likely to occur;

c) the case of a controlled system where most erroneous control signals are immediately "catastrophic", but the control software is engineered to detect its own internal errors and then issue a safe output and reset itself to a known state from which the program is likely to proceed correctly. One may then assume that most benign failures are due to this mechanism, and likely to be followed by successes: $0<p_{bc}<p_{sc}$;

d) the extreme of case c) above: the control software issues a safe output but after the reset takes place pointwise catastrophic failures cannot immediately happen ($p_{bc} = 0$).

The different situations illustrated can be summarised with the relation $p_{bc} = k \cdot p_{sc}$, where k is a non negative real number. We can thus determine the values of the transition probabilities as a function of k and of the distribution of p_i. In Section 4 we will discuss the effects of these scenarios (assigning proper values to k) on the probability of mission failure. Two cases ($k=0$ and $k>0$) have been distinguished for mathematical reasons and the resulting expressions are reported in Table 1. We detail here the derivation of the transition probabilities for the case k=0. In this case, $p_{bc}=0$, $p_{bb}=1$ and the system 3.1 reduces to:

$$\begin{cases} p_s = p_{ss} \cdot p_s + p_{b_1} + p_{ss} \cdot p_c \\ p_{b_1} = p_{sb} \cdot p_s \cdot p_1 + p_{b_2} + p_{sb} \cdot p_c \cdot p_1 \\ \ldots \\ p_{b_i} = p_{sb} \cdot p_s \cdot p_i + p_{b_{i+1}} + p_{sb} \cdot p_c \cdot p_i \quad i=2,3,\ldots \\ \ldots \\ p_c = p_{sc} \cdot p_s + p_{sc} \cdot p_c \\ p_s + \sum_i p_{b_i} + p_c = 1 \end{cases}$$

From the equation for p_c, the expression for p_{sc} is immediately derivable: $p_{sc} = \frac{p_c}{1-p_b}$

To obtain the expression for p_{sb} we first derive p_{b_1}:

$p_b = \sum_{i=1}^{\infty} p_{b_i} = p_{sb} \cdot (1-p_b) \cdot \sum_{i=1}^{\infty} p_i + \sum_{i=2}^{\infty} p_{b_i}$ which can be manipulated as:

$\sum_{i=1}^{\infty} p_{b_i} - \sum_{i=2}^{\infty} p_{b_i} = p_{sb} \cdot (1-p_b) \cdot \sum_{i=1}^{\infty} p_i$ from which $p_{b_1} = p_{sb} \cdot (1-p_b)$.

Then, from the equation for p_{b_i}, we get $p_{b_{i+1}} = p_{b_i} - p_{sb} \cdot (1 - p_b) \cdot p_i$; substituting p_{b_i} with its expression in terms of $p_{b_{i-1}}$ recursively up to p_{b_1} and using the expression for p_{b_1} just found:

$p_{b_{i+1}} = p_{sb} \cdot (1 - p_b) - p_{sb} \cdot (1 - p_b) \cdot p_1 - ... - p_{sb} \cdot (1 - p_b) \cdot p_i$ and then

$p_{b_{i+1}} = p_{sb} \cdot (1 - p_b) \cdot \left(1 - \sum_{j=1}^{i} p_j\right)$ that is $p_{b_{i+1}} = p_{sb} \cdot (1 - p_b) \cdot \sum_{j=i+1}^{\infty} p_j$.

Using the last expression in p_b:

$p_b = \sum_{i=1}^{\infty} p_{b_i} = \sum_{i=0}^{\infty} p_{b_{i+1}} = p_{sb} \cdot (1 - p_b) \cdot \sum_{i=0}^{\infty} \sum_{j=i+1}^{\infty} p_j$ that is

$p_b = p_{sb} \cdot (1 - p_b) \cdot \sum_{k=1}^{\infty} k \cdot p_k$, from which $p_{sb} = \dfrac{p_b}{(1 - p_b) \cdot \sum_{k=1}^{\infty} k \cdot p_k}$.

Once obtained the expressions for p_{sc} and p_{sb}, p_{ss} can be immediately derived:

$$p_{ss} = 1 - p_{sb} - p_{sc} = \frac{p_s \cdot \sum_{i=1}^{\infty} i \cdot p_i - p_b}{(1 - p_b) \cdot \sum_{i=1}^{\infty} i \cdot p_i}.$$

	k = 0	**k > 0**
p_{bb}	1	$\frac{1-p_{bb}}{k} = \frac{p_c - p_b \cdot (1-p_{bb})}{(1-p_b)} + \frac{p_b \cdot (1-p_{bb})^2 \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i}{(1-p_b) \cdot \left(1 - p_{bb} \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i\right)}$
p_{bc}	0	1-p_{bb}
p_{sb}	$\frac{p_b}{(1-p_b) \cdot \sum_{i=1}^{\infty} i \cdot p_i}$	$\frac{p_b \cdot (1-p_{bb})}{(1-p_b) \cdot \left(1 - p_{bb} \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i\right)}$
p_{sc}	$\frac{p_c}{1-p_b}$	$\frac{1-p_{bb}}{k}$
p_{ss}	$\frac{p_s \cdot \sum_{i=1}^{\infty} i \cdot p_i - p_b}{(1-p_b) \cdot \sum_{i=1}^{\infty} i \cdot p_i}$	$\frac{1}{1 - p_{bb} \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i} - \frac{p_c}{1-p_b} + \frac{(2 \cdot p_b \cdot p_{bb} - p_b - p_{bb}) \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i}{(1-p_b) \cdot \left(1 - p_{bb} \cdot \sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i\right)}$

Table 1. **Expressions for the transition probabilities in case of k = 0 and k > 0.**

Note that in the case of k > 0 an implicit expression for p_{bb} is given. So we have to solve numerically the equation for p_{bb} to obtain values for the other transition probabilities. The solution must refer to specific distributions of probability and is restricted to those distributions such that it is possible to give a finite expression equivalent to $\sum_{i=1}^{\infty} p_{bb}^{i-1} \cdot p_i$. For a number of distributions this expression is known 20 and we will show the numerical results for some of them.

3.3 The Proposed Model

The model representing the program in its operational context is shown in Figure 2.

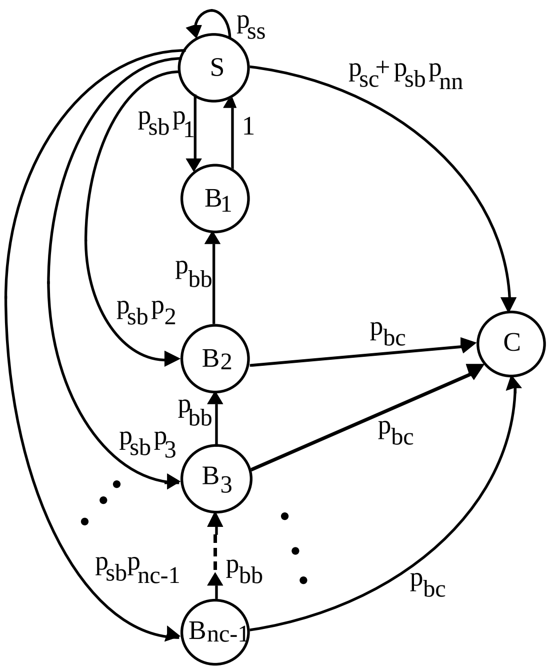

Figure 2. **The model for the program in its operational context.**

This model accounts for the two different characteristics we neglected in the previous model: sequences of n_c or more benign failures cause the controlled system to fail and missions last at most n iterations. Furthermore, a mission terminates successfully if no pointwise catastrophic failure or sequences of at least n_c benign failures are experienced, otherwise terminates with failure as soon as one of these two events is observed. Compared to the model of Figure 1, all states B_i, for $i \geq n_c$, disappeared and the term $p_{sb} \cdot p_{nn}$, where $p_{nn} = \sum_{i=n_c}^{\infty} p_i$, has been added to the arc from S to C to capture that sequences of benign failures longer than n_c -1 now lead to a mission failure. A simplification has been here introduced, since the exact modelling of sequences of n_c or more failures requires a sequence of n_c states (including C) instead of simply adding the probability $p_{sb} \cdot p_{nn}$ on the arc from S to C. However, the error introduced by such simplification is negligible, as better detailed in 21.

The values for the transition probabilities, derived from the model of Figure 1, can be applied and the model solved very easily. The solutions are expressed in terms of the parameters p_s, p_b, p_c, k, n_c and the distribution of the length of staying in a failure region. The parameter k is a characteristic of the considered software. With n_c-1 we represent the maximum number of consecutive benign failures that can happen without causing the mission to fail, i.e., the inertia of the system, thus it determines the resilience of the controlled system. Last the distribution function describes the length of stays in failure regions; it depends on the trajectories and the failure regions of the input space.

3.4 Probability of Surviving a Mission

The general expression for the probability of surviving a mission of a constant number n of iterations is the following:

$$P(\text{mission success}) = \prod_{i=1}^{n} P\left(X_i \neq C \mid X_{i-1} \neq C\right), \text{ where:}$$

$$P\left(X_i \neq C \mid X_{i-1} \neq C\right) = 1 - P\left(X_i = C \mid X_{i-1} \neq C\right) =$$

$$= 1 - \left(p_{sc} + p_{sb} \cdot p_{nn}\right) \cdot P\left(X_{i-1} = S \mid X_{i-1} \neq C\right) - p_{bc} \cdot P\left(X_{i-1} = B_2 \text{ or... or } X_{i-1} = B_{n_c - 1} \mid X_{i-1} \neq C\right)$$

It can be observed that the second and third terms in the last expression represent the probability of the occurrence of mission failure at the generic iteration. The second term collects the probability of mission failure after a success, due either to the occurrence of a pointwise catastrophic failure or to the entrance in a failure region in which the application remains for more than n_c-1 iterations. The third term is the probability of pointwise catastrophic failure after a benign failure (that is, while in a failure region).

4 Evaluations Results

In this section we analyse the model proposed in Section 3.2 in order to capture the effects determined by variations of the several parameters. We start by analysing the effects on reliability of different distribution functions of the length of stays in failure regions. Actually we use distribution functions belonging to different families (with the same values of p_{nn}, the probability of exceeding n_c-1 consecutive failures). For a better understanding, we also separate the contribution to the probability of mission failure due to serial failures and to pointwise catastrophic failures. Then we investigate on the changes due to the different structural characteristics of the software considered in the previous section by varying the parameter k. Next, we perform an analysis of the variations of the reliability as a function of the probability p_s obtained by testing, fixing all the other parameters and considering two different values for the ratio between p_b and p_c. In this way the robustness of our model against inaccurate assessment of the starting parameters can be checked. Last, we show the impact of varying the critical threshold n_c (representing the resilience of the controlled system). These analyses include, whenever appropriate, also the case of absence of correlation between successive inputs, to allow comparison with the independence assumption.

4.1 Distribution Functions, Parameters and their Assigned Values

The distributions we include in our analysis are some common distributions from the literature and some special limiting distributions, one of which can be shown to provide lower bounds on the dependability figures while the others are useful to explain some tendency. These distributions are:

- geometric distribution defined as: $p_i = q(1-q)^{i-1}, i \geq 1$ for $q \in (0,1)$ and $p_1 = 1$, $p_i = 0,\ i \geq 2$ for $q=0$;
- modified Poisson distribution defined as: $p_i = \dfrac{e^{-\alpha}\alpha^{i-1}}{(i-1)!}, i \geq 1,\ \alpha > 0$;
- modified negative binomial defined as: $p_i = \binom{i+r-2}{r-1} q^r (1-q)^{i-1}, i \geq 1,\ q \in (0,1)$ and $r \geq 1$ (we use r=5) or $p_1 = 1$, $p_i = 0,\ i \geq 2$ for $q=0$;

and, once a value for p_{nn} has been fixed,

- a distribution d1 defined as: $p_1 = (1 - p_{nn})$, $p_{n_c} = p_{nn}$ and $p_i = 0$ for $i \neq 1$ and $i \neq n_c$;
- a distribution d2 defined as: $p_1 = \frac{1 - p_{nn}}{2}$, $p_{n_c - 1} = \frac{1 - p_{nn}}{2}$, $p_{n_c} = p_{nn}$ and $p_i = 0$ for $i \neq 1$, $i \neq n_c\text{-}1$ and $i \neq n_c$;
- a distribution d3 defined as: $p_{n_c - 1} = 1 - p_{nn}$, $p_{n_c} = \frac{p_{nn}}{2}$, $p_{2n_c} = \frac{p_{nn}}{2}$ and $p_i = 0$ for $i \neq n_c\text{-}1$, $i \neq n_c$ and $i \neq 2n_c$.

The number of iterations in a mission, n, is 10^6 (a realistic number, e.g., for civil avionics where the average duration of one iteration could be 20-50 milliseconds and the mission duration could be around 10 hours). The meaning of the other parameters has already been described. Thus Table 2 reports the default values used for each parameter in those evaluations in which it is assumed as a constant; when variations of a given parameter are used, the variation range is explicitly indicated.

Parameter	**values**
$p_b = 5\ 10^{-5}$	$n_c = 10$
$p_c = 10^{-11}$	$n = 10^6$
$p_s = 1 - p_b - p_c$	$k = 100$
	$p_{nn} = 2\ 10^{-5}$

Table 2. **Parameter values used in the evaluation.**

4.2 Effects of Different Distributions of the Length of Stays in Failure Regions

In this subsection we evaluate the probability of mission failure obtained from the model as a function of the variation of the probability of exceeding a sequence of n_c-1 consecutive failures, p_{nn}. We use the six distributions previously described to model the length of stays in failure regions, a variation range from 0 to 5 10^{-5} for p_{nn} and the values shown in Table 2 for the other parameters. Figures 3 shows the probability of mission failure.

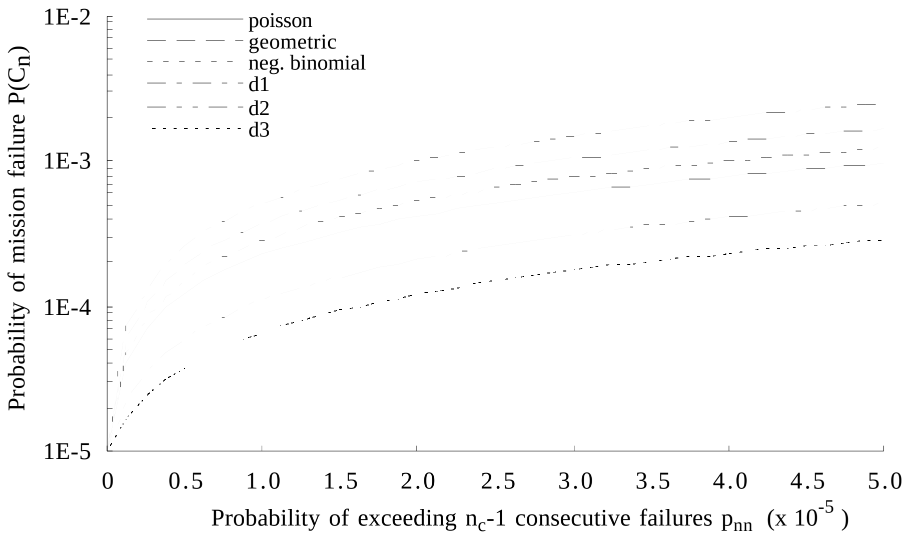

Figure 3. **Probability of mission failure as a function of p_{nn}.**

First it can be noticed that for all the distributions the probability of mission failure increases for increasing values of p_{nn}. This is obviously as expected: the higher the probability of encountering sequences longer than n_c-1 of benign failures, the higher is the number of mission failures. Second, the various distribution functions show different behaviours for the same value of p_{nn}. This difference seems to depend primarily on their respective mean: the reliability measures obtained for the various distributions are ranked in the same order as their means. d1 has the lowest mean and shows the worst behaviour among all the distributions while d2 and d3 show the best behaviour and have the highest mean.

To improve understanding, we separately analysed the probabilities of the two events contributing to the probability of mission failure $P(C_n)$, that is the probability of pointwise catastrophic failure (denoted with p_{point}) and the probability of occurrence of a series of more than n_c-1 benign failures (denoted with p_{serial}). p_{point} depends on p_{sc} and p_{bc}, while p_{serial} depends on $p_{sb} \cdot p_{nn}$. Since the figures obtained for the various distributions were quite similar among them, we show in Figure 4 the plots of p_{point}, p_{serial} and $P(C_n)$ for the geometric and the modified Poisson distributions, for the same setting of parameters as in Figure 3. To improve readability, only one curve has been shown for p_{point} because this quantity is nearly the same for the two considered distributions. Figure 4 shows that i) the value of p_{point} is nearly constant while p_{serial} varies for different distributions, and ii) p_{serial} constitutes the dominant contribution to $P(C_n)$ (apart for values close to zero of p_{nn} where the influence of p_{point} is dominant).

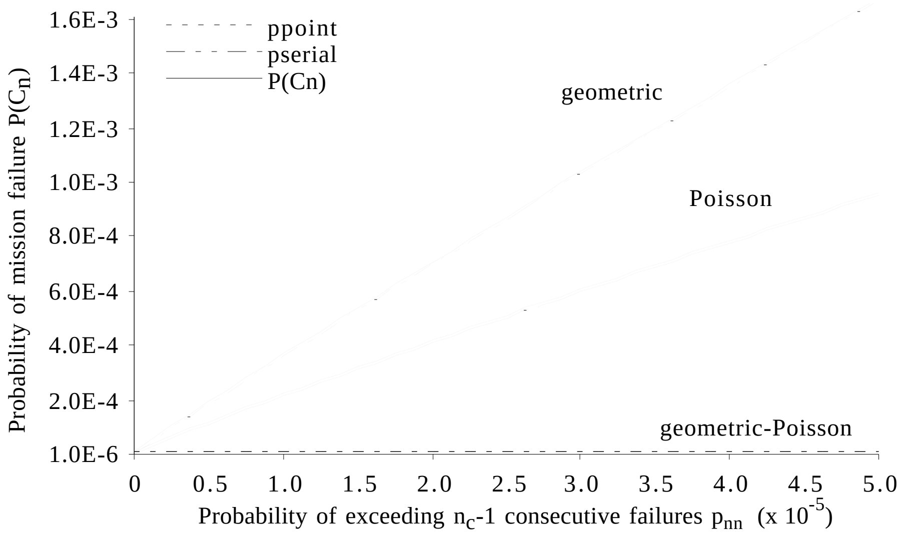

Figure 4. **p_{point} and p_{serial} as function of p_{nn}.**

Point i) above allows to explain the role of the mean of the distributions. Distributions with low mean represent input trajectories crossing failure regions more often but with a shorter permanence than those represented by distributions with higher mean. Thus, the lower the mean of a distribution, the higher becomes the corresponding probability p_{sb}. (For k =0, this result derives directly from Table 1, where p_{sb} is inversely proportional to the mean of the length of stays in a failure region.) Hence, having the same value for $p_{nn,}$ distributions with lower mean, i.e., higher values for p_{sb}, show higher p_{serial} and thus a worse behaviour.

4.3 Different Scenarios

In Section 3 we have introduced the relation $p_{bc} = k \cdot p_{sc}$ to solve the equation system 3.1 and explained how different scenarios can be modelled by assigning different values to the parameter k. The aim of this subsection is to investigate on the consequences of the scenario changes. We computed the probability of mission failure for different values of k (ranging from 0 to

10^7), with p_{nn} variable from 0 to 5 10^{-5} and the values in Table 2 for the other parameters. The distributions considered in this analysis have been the geometric, the modified negative binomial and the modified Poisson. The figures obtained for the various k were so close to each other that plotting them would have been useless. So, we decided to show in Figure 5, for each distribution, the plot of the difference on the probability of mission failure between the highest (where k =0) and the lowest (where k =10^7) values obtained.

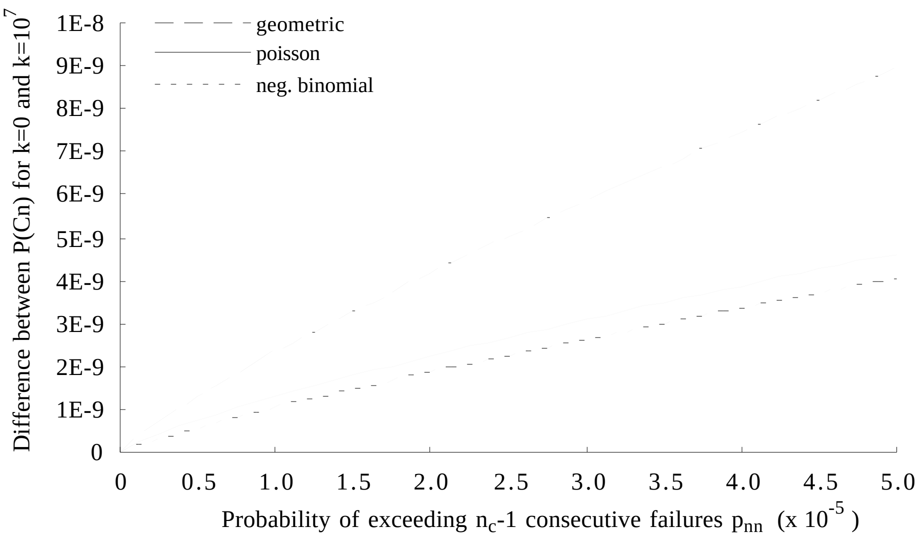

Figure 5. **Differences in the probability of mission failure for k = 0 and k = 10^7.**

It can be observed that for all the three distributions the maximum difference is extremely low, being at most of the order of 10^{-8} against absolute values of the order of 10^{-3} (see Figure 3). Hence, we can conclude that the changes of k do not affect the total probability of mission failure, that is the reliability is practically insensitive w.r.t. scenario changes.

4.4 Robustness of the Model

Besides the possibility or easiness to obtain proper assessments of the parameters, another important factor to determine the effective utility of a model is its robustness against inaccurate values assigned to some parameter. Here we analyse the reliability variations as a function of the probability p_s, fixing all the other parameters and considering two different values for the ratio between p_b and p_c. Figures 6 and 7 show respectively the probability of mission failure for $p_b = 10^6\ p_c$ and $p_b = 10^4\ p_c$, varying 1-p_s from 10^{-6} to 10^{-4}. In the Figures we included also the probability of mission failure in the case of independence between successive inputs. In this case, we have $p_{nn} = p_b^{n_c} / (1 - p_b)$ and a very low probability of incurring in a sequence of benign failures equal or longer than n_c. Due to the irrelevant contribution to $P(C_n)$ given by p_{serial} the case of independence can be therefore considered a lower bound for the probability of mission failure that can be expected given p_s, p_b and p_c.

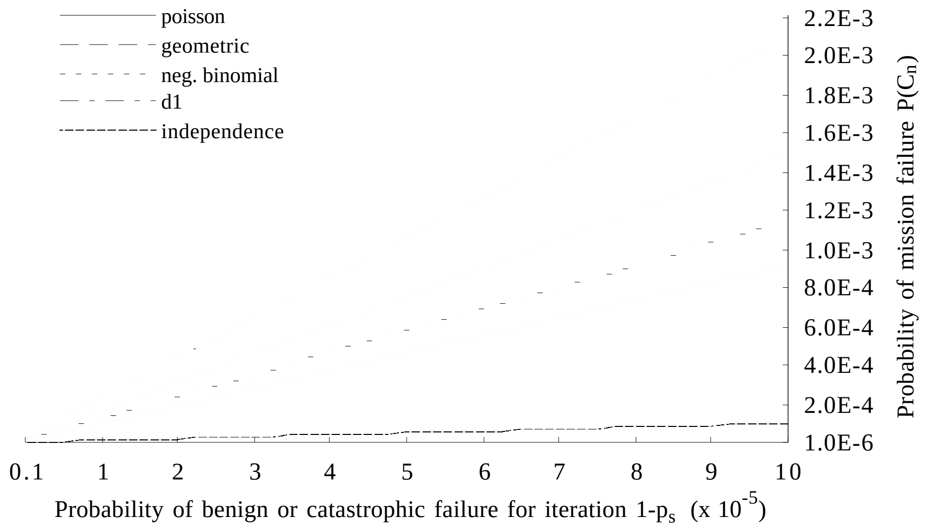

Figure 6. **Probability of mission failure as a function of p_s, with $p_b = 10^6\ p_c$.**

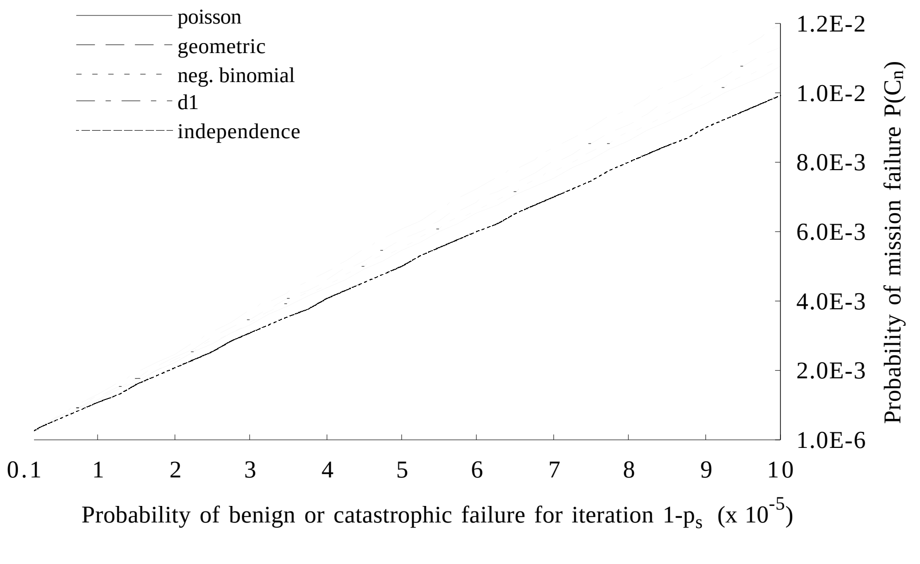

Figure 7. **Probability of mission failure as a function of p_s, with $p_b = 10^4\ p_c$.**

A few considerations with respect to the sensitivity to p_s follow. First it can be noticed that, in the entire range we considered, the error in estimating $P(C_n)$ because of a wrong value assigned to p_s depends only on how inaccurate the estimate is. There are no critical subsets in the range in which the inaccuracy is particularly dangerous. For the same ratio between p_b and p_c the curves show that an error in the estimate of p_s (say of 10%) determines an error of the same order of magnitude in the expectation of $P(C_n)$. In Figure 7, one can appreciate that this holds also for the case of independence. Last, it can be noticed that lowering the ratio between p_b and p_c, i.e., passing from $p_b = 10^6\ p_c$ to $p_b = 10^4\ p_c$, the probability of mission failure increases, and the curves show bit higher slopes: $P(C_n)$ is a bit more sensitive to variations of p_s.

Furthermore, the curves in Figure 7 are much closer to each other and are ranked in the same order than those in Figure 6. Actually the absolute distance between any two curves for the same value of p_s is almost the same both in Figure 6 and in Figure 7. To explain this one must notice that the values for p_c change of orders of magnitude and this significantly increases the probability of pointwise catastrophic failures while those for p_b remain approximately the same: leaving p_{nn} unchanged for each distribution the same contribution to mission failure due to sequences of benign failures is obtained.

4.5 Resilience of the Controlled System

Last we show the effect of varying n_c. This analysis is interesting for software suitable for several physical systems; different values of n_c represent the different inertia of the controlled systems. We perform an evaluation assuming that, changing the environment, the distribution function of the length of stays in a failure region remains unchanged. Figure 8 shows the probability of mission failure in a logarithmic scale for the geometric, modified Poisson, modified negative binomial distributions and for the case of independence between successive inputs. The used distribution functions are fixed in accordance with the values reported in Table 2. The range chosen for n_c extends from 2 to 16 (for values higher than 16 the probability of mission failure does not show sensible variations).

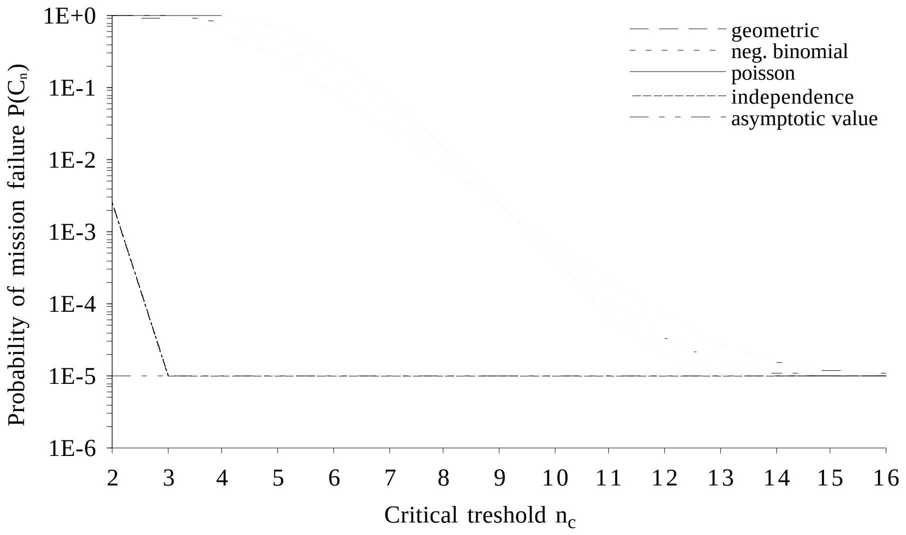

Figure 8. **Probability of mission failure as function of n_c for given distributions.**

As expected, growing values of n_c imply a decrease of the value of p_{nn} and, consequently, of the probability of mission failure. For all the distribution functions considered, when p_{nn} approaches zero, the probability of mission failure converge to the same value, which is the probability of mission failure due to pointwise catastrophic failures. Obviously, the independence curve converge for a very low value of n_c compared to the other distributions (3 against 16, for our parameters setting).

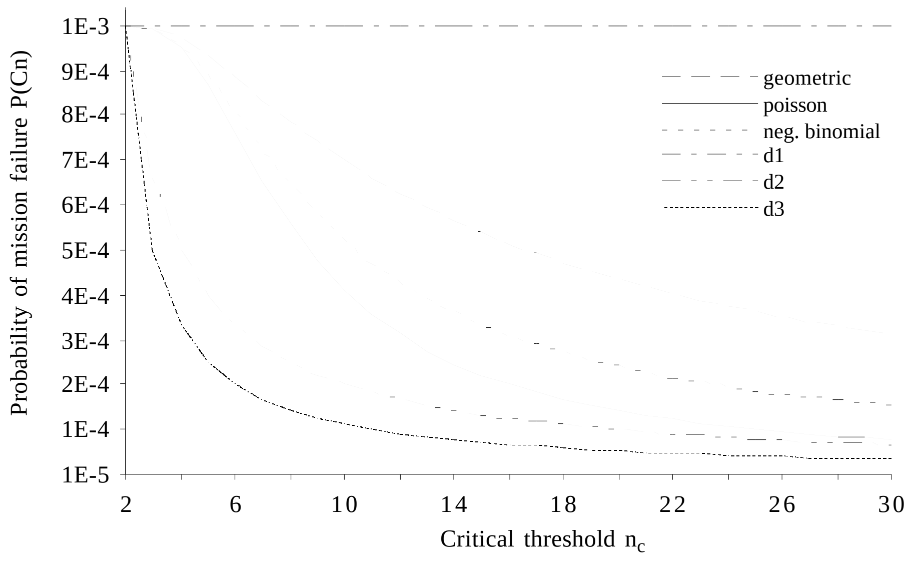

Figure 9. **Probability of mission failure as a function of n_c (with fixed p_{nn}).**

To complete our analysis on the effects of variations of n_c, an alternative evaluation is performed. The value for p_{nn} is fixed (as reported in Table 2) so changes of n_c imply changes in the parameters of the distribution functions. In Figure 9 we show the probability of mission failure with n_c varying from 2 to 30, considering all the distribution functions reported in the Subsection 4.1. Fixing the value of p_{nn}, the variation of n_c only affects the mean of the length of stays in a failure region. Increasing values of n_c imply higher values of the mean of these distributions. As a consequence, reliability improves, as already discussed in Subsection 4.2.

5 Conclusions

In this paper we offered two contributions to structural models for predicting the dependability of iterative software that account for both dependencies between input values of successive iterations and the possibility that repeated, non fatal, failures may together cause mission failure. The first concerns the problem of providing accurate estimates for the basic parameters of models. We extended the previous work by proposing a modelling framework and a procedure for providing accurate estimates of the basic parameters of the models, thus addressing the problem of the real usability of analytical models. The models developed use as basic knowledge the steady state probabilities of the software system in a context where two characteristics of the original environment are relaxed, namely; the possibility for repeated benign failures to cause a catastrophic failure is not considered and the missions have not a fixed duration, but after a catastrophic failure the software is reset to the initial state. In this context, steady state probabilities appear to be relatively easy and cheap to determine if compared with the difficulty to assess the system dependability indicators of interest. In this way, an interesting compromise is reached between a fairly realistic model for obtaining predictions of dependability attributes of a system and difficulty (and costs) in obtaining the basic knowledge necessary to resolve the model.

The second contribution consists of extensive analyses performed to investigate on the effects of variations of the various parameters on the reliability of iterative software. We analysed the sensitivity of the model to the correlation between successive inputs, the different structural characteristics of the system at hand and to our starting parameters in order to check the robustness of the model against inaccurate initial assessments.

Acknowledgements

S. La Torre was supported by a C.N.R. (Consiglio Nazionale delle Ricerche) fellowship.

References

1 Arlat, J., Kanoun, K. and Laprie, J. C., “Dependability Modelling and Evaluation of Software Fault-Tolerant Systems,” IEEE Transactions on Computers, Vol. C-39, 1990, pp. 504-512.

2 Tai, A. T., Avizienis, A. and Meyer, J. F., “Evaluation of Fault Tolerant Software: a Performability Modeling Approach,” C. E. Landwher, B. Randell and L. Simoncini, Springer-Verlag, 1992, pp. 113-135

3 Tai, A. T., Avizienis, A. and Meyer, J. F., “Performability Enhancement of Fault-Tolerant Software,” IEEE Transactions on Reliability, Vol. R-42, 1993, pp. 227-237.

4 Chiaradonna, S., Bondavalli, A. and Strigini, L., “On Performability Modeling and Evaluation of Software Fault Tolerance Structures,” in Proc. EDCC1, Berlin, Germany, 1994, pp. 97-114.

5 Amman, P. E. and Knight, J. C., “Data Diversity: An Approach to Software Fault Tolerance,” IEEE Transactions on Computers, Vol. C-37, 1988, pp. 418-425.

6 Bishop, P. G. and Pullen, F. D., “PODS Revisited - A Study of Software Failure Behaviour,” in Proc. 18th International Symposium on Fault-Tolerant Computing (FTCS-18), Tokyo, Japan, 1988, pp. 1-8.

7 Bishop, P. G., “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 International Symposium on Fault-Tolerant Computing (FTCS-23), Toulouse, France, 1993, pp. 98-107.

8 Bondavalli, A., Chiaradonna, S., Di Giandomenico, F. and Strigini, L., “Dependability Models for Iterative Software Considering Correlation between Successive Inputs,” in Proc. IEEE Int. Conference on Performance and Dependability, Erlangen, Germany, 1995, pp. 13-21.

9 Csenski, A., “Recovery Block Reliability Analysis with Failure Clustering,” in Proc. 1st IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-1), Santa Barbara, California, 1991, pp.

10 Tomek, L. A., Muppala, J. K. and Trivedi, K. S., “Modeling Correlation in Software Recovery Blocks,” IEEE Transactions on Software Engineering, Vol. SE-19, 1993, pp. 1071-1085.

11 Randell, B., “System Structure for Software Fault Tolerance,” IEEE Transactions on Software Engineering, Vol. SE-1, 1975, pp. 220-232.

12 Bertolino, A., “Software Testing for Dependability Assessment,” in Proc. Objective Quality: Second Symposium on Software Quality Techniques and Acquisition Criteria, Florence, Italy, 1995, pp. 236-248.

13 Bertolino, A. and Strigini, L., “On the Use of Testability Measures for Dependability Assessment,” IEEE Transactions on Software Engineering, Vol. SE-22, 1996, pp.

14 Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E. and Powell, D., “Fault Injection for Dependability Validation - A Methodology and Some

Applications," IEEE Transactions on Software Engineering, Vol. SE-16, 1990, pp. 166-182.

15 Madeira, H. and Silva, J. G., "Experimental Evaluation of the Fail-Silent Behaviour in Computers without Error Masking," in Proc. 24th Interational Symposium on Fault Tolerant Computing Systems (FTCS-24), Austin, Texas, 1994, pp. 350-359.

16 Arlat, J., Costes, A., Crouzet, Y., Laprie, J.-C. and Powell, D., "Fault Injection and Dependability Evaluation of Fault-tolerant Systems," IEEE Transactions on Computers, Vol. C-42, 1993, pp. 913-923.

17 Lyu, M. R. and He, Y., "Improving the N-Version Programming Process Through the Evolution of a Design Paradigm," IEEE Transactions on Reliability, Sp. Issue on Fault Tolerant Software, Vol. R-42, 1993, pp. 179-189.

18 Meyer, J. F., "On Evaluating the Performability of Degradable Computing Systems," IEEE Transactions on Computers, Vol. C-29, 1980, pp. 720-731.

19 La Torre, S., Chiaradonna, S., Di Giandomenico, F. and Bondavalli, A., "The Effects of Input Correlation on the Dependability of Iterative Software," IEI-CNR, Pisa, Italy Internal Report, No. B4-24, May, 1995.

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

21 Bondavalli, A., Chiaradonna, S., Di Giandomenico, F. and Strigini, L., "Modelling Correlation among Successive Inputs in Software Dependability Analyses," CNUCE/CNR Technical Report, C94-20, 1994.