X-ware reliability and availability modeling

Jean-Claude Laprie, Karama Kanoun

X-Ware Reliability and Availability Modeling*

Jean-Claude Laprie, Karama Kanoun

LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex — France

Abstract

This paper addresses the problem of modeling a system's reliability and availability with respect to the various classes of faults (physical and design, internal and external) which may affect the service delivered to its users. Hardware-and-software models are currently exceptions in spite of the users requirements; these requirements are expressed in terms of failures independently of their sources, i.e. the various classes of faults. The causes of this situation are analyzed; it is shown that there is no theoretical impediment to deriving such models, and that the classical reliability theory can be generalized in order to cover both hardware and software viewpoints that is X-Ware. After a first section, the introduction, which summarizes the current state-of-the-art with regard to the dependability requirements from the users' viewpoint, the body of the paper is composed of two sections. The second section is devoted to system behavior up to failure; it focuses on failure rate and reliability, considering in turn atomic (or single component) systems, and systems made out of components. The third section deals with the sequence of failures when considering restoration actions consecutive to maintenance; failure intensity, reliability and availability are derived for various forms of maintenance.

Index terms

Hardware reliability and availability modeling

Software reliability and availability modeling

Reliability growth

Renewal processes

Non stationary stochastic processes

System structure models

---

* This work has been partially supported by the CEC under ESPRIT BRA Project 3092, "Predictably Dependable Computing Systems" (PDCS).

1 Introduction

When dealing with the assessment of dependability (a concept encompassing reliability, availability, safety, etc. [43, 7, 46]) the users of computing systems are interested in obtaining figures resulting from modeling and evaluation of *systems*, composed of hardware and software, with respect to both *physical and design faults*. Supports to this statement may be found when considering:

- the requirements for systems in terms of dependability; an example is provided by the requirements for electronic switching systems (figure 1), in which it is explicitly stated that they apply to both hardware and software, especially in terms of reliability and availability;
- the sources of failure of computing systems; an example is provided in figure 2, which gives the results of a survey regarding the sources of unavailability for electronic switching systems and for transaction processing systems; clearly, an evaluation which would be performed with respect to hardware failures only would not be representative of the actual behavior of the considered systems.

The hardware and software for switching systems must be designed to meet the requirements shown in the table below

Operation	• Continuous (20 years)
Time Shared Channels	• Thousands
Recovery Time	• Critical
Value of Reliability	• High 1 call per 100,000 call cut off
Downtime	• < 3 minutes / yr
Synchronization	• Network Sync
System Growth Data Base Changes Program Changes Maintenance	All must be done with the system on-line and operational

Figure 1 - Requirements for ESSs [17]

UNAVAILABILITY SOURCES	ELECTRONIC SWITCHING	TRANSACTION PROCESSING
Hardware failures	20 %	40 %
Environment		5 %
Software failures	15 %	30 %
Recovery deficiencies	35 %	
Incorrect procedures	30 %	20 %
Others, such as updating		5 %

Figure 2 - Sources of failures [74]

In addition, it is worth mentioning that agencies are becoming aware of the necessity of performing evaluations encompassing both hardware and software. An example is given by figure 3, which is extracted from an invitation to tender from ESA, the European Space Agency [24].

"Studies have been conducted for ESA into software reliability requirements for ESA Space programmes. These studies, conducted in 1986, concluded:

a) Software failure is a process that appears to the observer to be "random", therefore the term "reliability" is meaningful when applied to a system which includes software and the process can be modeled as stochastic.

b) The definition of "Reliability" is identical for a system which includes software as it is for a purely hardware system. It is the classical definition: "the probability of successful operation for a given period under given conditions".

c) The specification of numerical levels of reliability for a complete system is meaningless unless the reliability of the software which it contains is similarly quantified and the level of reliability achieved by that software is verified."

Figure 3 - Statement from ESA

Faced with these user requirements, hardware-and-software evaluations are far from being current practice, with a few exceptions (see e.g. [67, 18, 6, 66, 71]). An explanation to this state of affairs

lies in the fact that hardware evaluation and software evaluation have followed courses which can hardly be more separated.

Hardware evaluation has concentrated on operational life, focusing on the influence on dependability of the system structure. A number of significant results have been derived:

- the role and influence of the coverage of the fault-tolerance strategies and mechanisms [13, 4],
- construction [10] and processing [29, 12] of large, stiff, Markov models,
- definition and evaluation of performance-related measures of dependability, which are usually gathered in the concept of performability [56, 70].

These results have been integrated in software packages, such as ARIES, HARP, SAVE, SURF, etc.

However, as far as elementary data are concerned, reliance has generally been placed on data bases such as the MIL-HDBK-217 which are limited to permanent faults, whereas it is currently admitted that temporary faults constitute a major source of failure. In addition, it has largely been ignored that the reliability of hardware parts is significantly growing during the whole system's life, as shown, for example, by the experimental data displayed in [9].

Software evaluation has mainly concentrated on the development phase, focusing on the reliability growth of single-component ("black box") systems. Many models have been proposed (see surveys such as [77, 66, 54]). As these models are aimed at predicting the future reliability from the failure data accumulated in the past, attention has been specifically devoted to the prediction problem [54]. Less attention has been paid to accounting for the structure of a software system, where most of the corresponding work has been restricted to the failure process, either for non-fault tolerant [52, 42] or for fault-tolerant software systems [30, 28, 42, 3]; our recent work only [48] deals with evaluating the reliability growth of a system from the reliability growth of its components.

From the practical utilization viewpoint, the situation can be summarized as follows:

- hardware evaluation is relatively well incorporated in the design process, although the estimations are usually carried out more as an adjunct to the design methodology than an integral part of producing an optimised design.
- in the vast majority of software developments, evaluation is extremely limited if any; most of the published work devoted to applications on real data is in fact "post mortem" work; among the causes of this situation [35], it is worth mentioning:
  - the unrealistic hypotheses of most of the models,
  - the inability of the models to predict what will happen after a change either in the specification or after a transition between two phases of the system's life (e.g. from development-validation to operation),
  - the initial "over-selling" of the models: many authors have exaggerated their model's utilization possibilities as well as the lessons which can be drawn from this utilization.

This paper elaborates on previous work [44, 45, 48]. It is aimed at showing that, using deliberately simple mathematics, *the classical reliability theory can be extended in order to be interpreted from both hardware and software viewpoints*, what is termed in the sequel "X-ware". It will be shown that, even though the action mechanisms of the various classes of faults may be different from a physical viewpoint according to their causes, a single formulation can be used from the reliability modeling and statistical estimation viewpoints. The second section is devoted to the failure behavior of an X-ware system without taking into account the effect of restoration actions (the quantities of interest are thus the time to next failure or the associated failure rate), considering in turn atomic ("black box") systems, and systems made out of components. In the third section, we deal

with the behavior of an X-ware system with service restoration: it is devoted to the characterization of the sequence of the times to failure (i. e. the failure process); the measures of interest are thus the failure intensity and the availability. Non-redundant systems only are considered throughout this paper.

2 Failure behavior of an X-ware system

The first sub-section characterizes the behavior of atomic systems: discrete and continuous-time reliability expressions are derived. The behavior of systems made out of components is addressed in the second sub-section where structural models of a system according to different types of relations are first derived, which enable a precise definition of the notion of interpreter; behavior of single-interpreter and of multi-interpreter systems are then successively considered.

2.1 Atomic systems

The simplest functional model of a system is to see it as performing a mapping of its input domain I into its output space O.

An execution run of the system consists in selecting a sequence of input points. A trajectory in the input domain — not necessarily composed of contiguous points — can be associated with such a sequence. Each element in I is mapped to a unique element in O if we assume that the state variables are considered as part of I and/or O.

A system failure may result from [46]:

- the *activation* of a fault *internal* to the system, previously *dormant*; an internal fault may:
  - result from the physical failure of a hardware component; such faults are usually referred to as *physical faults*,
  - be a *design fault* affecting the software or the hardware;
- the *occurrence* of an *external* fault, originating from either the *physical* or the *human* environment of the system.

Two subspaces in the input space can thus be identified:

- $\mathbf{I_{fi}}$, the subspace of the faulty inputs,
- $\mathbf{I_{af}}$ the subspace of the inputs activating internal faults.

The *failure domain* of the system is $\mathbf{I_F} = \mathbf{I_{fi}} \cup \mathbf{I_{af}}$. When the input trajectory meets $\mathbf{I_F}$, an error occurs, which leads to failure.

At each selection of an input point, there is thus a non zero probability for the system to fail. Let p be this probability, assumed for the moment identical whatever the input point selected:

$$p = P\{\text{system failure at an input point selection} \mid \text{no failure at the previous input point selections}\}$$

Let $R_d(k)$ be the probability of no system failure during an execution run comprising k input point selections. We then have:

$$R_d(k) = (1 - p)^k \tag{2.1}$$

$R_d(k)$ is the *discrete-time system reliability*.

Let t_e be the execution duration associated with an input selection; t_e is supposed for the moment identical whatever the input point selected. Let t denote the time elapsed since the start of execution:

$$t = k\ t_e \tag{2.2}$$

The notion of (isolated) input point is not very well suited for a number of situations, such as control systems, executive software, hardware. Thus, let us assume that there exists a finite limit for p/t_e when t_e becomes vanishingly small. Let λ be this limit:

$$\lambda = \lim_{t_e \to 0} \frac{p}{t_e} \tag{2.3}$$

As the limit of the geometric distribution is the exponential distribution, we get:

$$R(t) = \lim_{t_e \to 0} R_d(k) = \exp(-\lambda t) \tag{2.4}$$

R(t) is the *continuous-time system reliability*, and λ is its failure rate.

Let us now relax the identity assumption of p and t_e with respect to the input point selections; let the following be defined:

$p(j) = P\{$ system failure at the jth input point selection | no failure at the previous input point selections$\}$

$t_e(j)$ = execution duration associated with the jth input selection

Relations (2.1) to (2.5) become:

$$R_d(k) = \prod_{j=1}^{k} [\, 1 - p(j)] \tag{2.5}$$

$$t = \sum_{j=1}^{k} t_e(j)$$

$$\lambda(j) = \lim_{\forall j\, t_e(j) \to 0} \frac{p(j)}{t_e(j)}$$

$$R_d(k) = \prod_{j=1}^{k} \{\, 1 - \lambda(j)\, t_e(j) + o[t_e(j)] \,\}$$

$$R(t) = \lim_{\forall j\, t_e(j) \to 0} R_d(k) = \exp\left(- \int_0^t \lambda(\tau)\, d\tau \right) \tag{2.6}$$

Relation (2.6) is nothing else than the general expression of a system's reliability: the above derivations are somewhat classical, they have however been introduced for the sake of completeness.

It could be argued that the above formulations are in fact better adapted to design faults or to external faults than to physical faults, since they are based on the existence of a failure domain, which may be felt as not existing with respect to physical faults as long as no hardware component fails. It should be remembered that, from a physics reliability viewpoint, there is no "sudden, unpredictable" failure. In fact, a hardware failure is due to anomalies (errors) at the electronic level, and these anomalies are caused by physico-chemical defects (faults). Stated in other terms, there are no fault-free systems – either hardware or software, there are only systems which have not yet failed. However, the notion of operational fault, i.e. which develops during system operation, and was thus not existing at the beginning of operational life is — although arbitrary [41] — a convenient and usual notion. Incorporating the notion of operational fault in the previous formulation can be conducted as follows. Let j_0 be the number of input point selections such that $p(j) = 0$ for $j < j_0$, $p(j) = p$ for $j \geq j_0$, and $u(j_0)$ the associated probability:

$$u(j_0) = P\{\, p(j) = 0 \;\; j < j_0 \,,\, p(j) = p \;\; j \geq j_0 \,\}$$

The expression of the discrete-time reliability becomes:

$$R_d(k) = \sum_{j_0=0}^{k} (1 - p)^{k-j_0} u(j_0) \quad , \text{ with } \quad \sum_{j_0=0}^{k} u(j_0) = 1$$

Going through the same steps as before, we get:

$$R(t) = \lim_{t_e \to 0} R_d(k) = \exp(-\lambda t)$$

What precedes shows that, although the action mechanisms of the various classes of faults may be different from a physical viewpoint according to their causes, a single formulation can be used from a probability modeling viewpoint. This formulation applies whatever the fault class considered, either internal or external, either physical or design-induced. In the case of software, the randomness comes, at least, from the trajectory in the input space which will activate the fault(s) [52]. In addition, it is now known that most of the software faults which are still present in operation, after validation, are "soft" faults, in the sense that their activation conditions are extremely difficult to reproduce, hence the difficulty of diagnosing and removing them[1], which adds to the randomness.

The constancy of the conditional failure probability at execution with respect to the execution sequence is directly related to the constancy of the failure rate with respect to time, as evidenced by relations (2.1), (2.4), (2.5), (2.6). The meaning is that the points of an input trajectory are not correlated *with respect to the failure process*. This statement is all the more likely to be true if the failure probability is low, i.e. if the quality of the system is high, and thus applies more to systems in operational life than to systems under development and validation. It is however an abstraction, and the question which immediately arises is: how well does this abstraction reflect reality?

As far as hardware is concerned, it has been shown for a long time (see e.g. [14]) that the failure rates of electronic components as estimated from experimental data actually decrease with time — even after the period of infant mortality. However, the decrease is generally low enough in order to be neglected. Going further, the interpretation of the failure data for satellites as described in [31] establishes a distinction between a) stable operating conditions where the failure rate is slowly decreasing — a constant failure rate is a satisfactorily assumption, and b) varying operating conditions which lead to failure rates which are significantly decreasing with time.

Similar phenomena have been noticed concerning software:

- constant failure rates for given, stable, operating conditions [63],
- high influence of system load [15].

In addition, a series of experimental studies conducted on computing systems, have confirmed the significant influence of the system load on both hardware and software failure processes [32].

The influence of varying operating conditions can be introduced by considering that *both the input trajectory and the failure domain* I_F *may be subject to variations*. The variation of the failure domain deserves some comments; it may be consecutive to two phenomena [32]:

1) Accumulation of physical faults which remain dormant under low load conditions, and are progressively activated as the load increases [57].
2) Creation of temporary faults resulting from the presence of rarely occurring combinations of conditions; examples are a) "pattern sensitive" faults in semiconductor memories, change in parameters of a hardware component (effect of temperature variation, delay in timing due to parasitic capacitance, etc.), or b) situations occurring when system load rises beyond a

[1] As an example, a large survey conducted on Tandem systems [27] concluded that, from the examination of several dozens of spooler error logs, only one software fault out of 132 was not a soft fault.

certain level, such as marginal timing and synchronization; the latter situation may affect software as well as hardware: the notion of temporary faults — especially intermittent faults — applies also to software [27]. Experimental work [58] has shown that the failure rates relative to temporary faults decrease significantly with time.

From a probabilistic viewpoint, the failure probability at execution in discrete time (resp. the failure rate in continuous time) may be considered as a random variable. The system reliability then results from the mixture of two distributions:

- distribution of the number of non-failed executions in given operating conditions, thus with a given, constant, failure probability at execution (resp. of the time to failure with a given, constant, failure rate),
- distribution of the probability of failure at execution (resp. of the failure rate).

Let $g_d(p)$ (resp. $g_d(\lambda)$) be the probability density function of the distribution of the probability of failure at execution (resp. of the failure rate), which may take G values relative to the realizations p_i, i=1,...,G, of p (resp. λ_i, i=1,...,G, of λ).

The expression of the discrete-time reliability becomes:

$$R_d(k) = \sum_{i=1}^{G} (1-p_i)^k \, g_d(p_i)$$

Going through similar steps as before, we get:

$$t = k \sum_{i=1}^{G} t_{ei} \, g_d(p_i) \qquad \lambda_i = \lim_{t_{ei} \to 0} \frac{p_i}{t_{ei}} \;, \; i=1,\ldots,G$$

where t_{ei} and λ_i are respectively the execution time and the failure rate for executions carried out under operating condition i, i=1,...,G. It comes:

$$R(t) = \lim_{\forall i \; t_{ei} \to 0} R_d(k) = \sum_{i=1}^{G} g_d(\lambda_i) \exp(-\lambda_i t)$$

This is the mixed exponential distribution.

When p is a continuous random variable with density function $g_c(p)$ (resp. λ is a continuous random variable with density function $g_c(\lambda)$), we get:

$$R_d(k) = \int_0^1 (1-p)^k \, g_c(p) \, dp \qquad R(t) = \int_0^\infty \exp(-\lambda t) \, g_c(\lambda) \, d\lambda$$

From the properties of the mixture of distributions (see e.g. [8]), the system failure rate is non increasing with time, whatever the distributions $g_d(.)$ and $g_c(.)$. For instance, a continuous distribution of wide generality is the Gamma distribution; in the case where λ is considered as a continuous random variable, we have:

$$g_c(\lambda) = \frac{b^a \lambda^{a-1} \exp(-b\lambda)}{\Gamma(a)} \;, \text{ and: } R(t) = \left(\frac{b}{t+b}\right)^a \;, \text{ which is a Pareto distribution.}$$

A model is of no use without data. That's where statistics come into play. Let M instances of the system be considered, executed independently. The term "independently" is a keyword for what follows. This is a classical assumption with respect to physical faults when several sets of hardware are run in parallel, supplied with the same input pattern sequences. Such an approach cannot

obviously be transposed to software. Independency of the executions of several systems means that they are supplied with *independent input sequences*. It is noteworthy that this reflects operational conditions when considering a base of deployed systems; for instance, the input sequences supplied to the same text processing software by individuals utilizing it in different locations, and for activities of completely different natures, are likely to exhibit independence with respect to residual fault activation.

Let M(k) (resp. M(t)) be the number of non-failed instances after k executions (resp. after an elapsed time of t since the start of the experiment) of each instance, M(0) = M; an instance failing at the jth execution, j=1,...,k (resp. at time τ, $\tau \in [0,t]$) is no longer executed. Independency of the executions of the various instances enable these executions to be considered as Bernoulli trials, and the discrete-time reliability is thus $R_d(k) = \frac{E[M(k)]}{M(0)}$ (resp. the continuous-time reliability is $R(t) = \frac{E[M(t)]}{M(0)}$). These relations are nothing other than the basic relations for the statistical interpretation of reliability as stated in the "general" systems reliability theory (e.g. [69, 40]). Statistical estimators of $R_d(k)$ and R(t) are then: $\hat{R}_d(k) = \frac{M(k)}{M(0)}$ and $\hat{R}(t) = \frac{M(t)}{M(0)}$.

What precedes shows that the relations forming the core of the statistical estimation of reliability for a set of hardware systems apply equally to software systems, *provided that the experimental conditions be in agreement with the underlying assumptions.*

2.2 Systems made out of components

2.2.1 System models

Adopting the spirit of [2], a system may be viewed from a structural viewpoint as a set of components bound together in order to interact; a component is itself a system, decomposed (again) into components, etc. The recursion stops when a system is considered as being *atomic*: any further internal structure cannot be discerned, or is of no interest and can be ignored. The model corresponding to the relation *is composed of* is a tree, whose nodes are the components; the levels of a tree obviously constitute a hierarchy[2]. Such a model does not enable the interactions between the components to be represented: the presence of arcs in a graphic representation would only be the materialization of the relation "is composed of" existing between a node and the set of its immediate successors. The set of the elements of a level of the tree gives only the list of the system components, with more or less *detail* according to the considered tree level: the lower the level, the more detailed is the list. In order to have a more representative view of the system, it is necessary to represent the relations existing between the components, i.e. through interaction diagrams where the nodes are the system components and the arcs the materialization of the existence of a common interface: an arc exists when two elements *can* interact. Although the relation "interacts with" is an essential relation when describing a system, it obviously does not enable any hierarchy to be inferred.

The use of the relations which have been considered up to now in modeling a system is illustrated by figure 4 in the case of a — deliberately — very simple system, where the components S_1, S_2, and S_3 can be, for instance, the application software, the executive software, and the hardware.

---

[2] As stressed in [65], it is essential when dealing with hierarchies a) to clearly identify the relation with respect to which hierarchical modeling is being done, and b) this relation has to be such that the directed graph representing it has no loops. Both conditions are fulfilled by the hierarchies which will be considered.

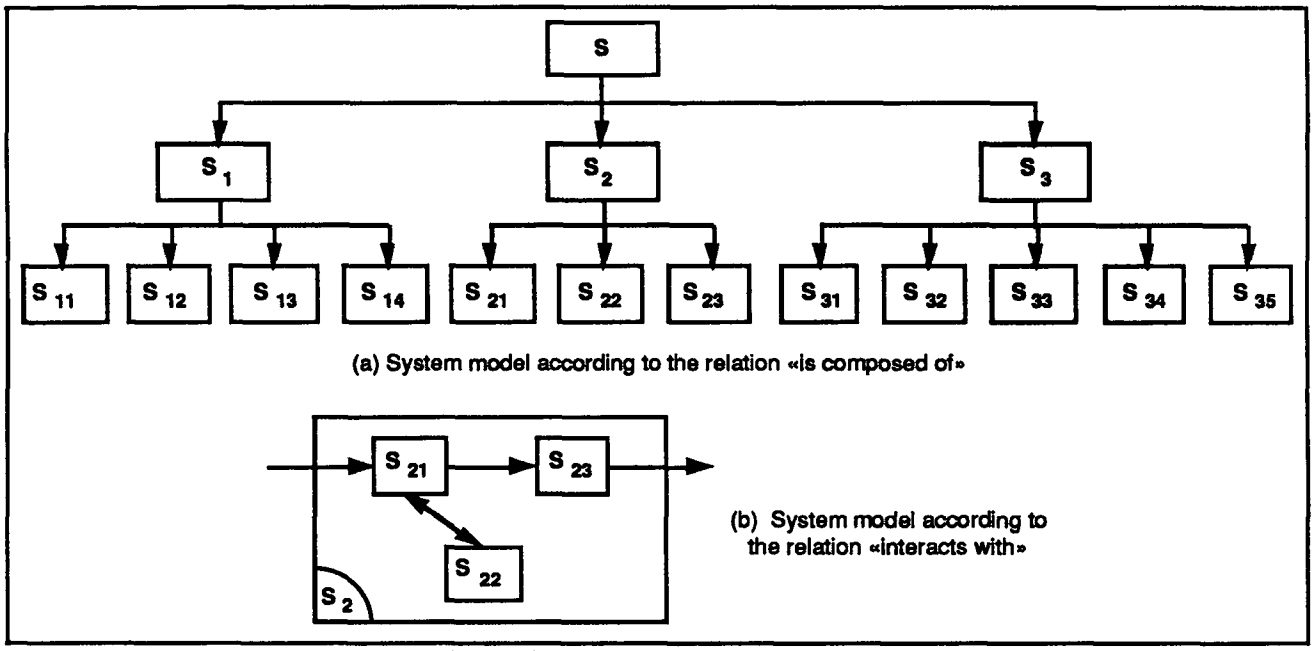

Figure 4 - Components of a system

The respective roles of a system and of its user(s) with respect to the notion of service are fixed: the system is the producer of the service, and the user is the consumer of the service. Therefore, there exists a natural hierarchy between the system and its user: the user *uses the service of* (or *delivered by*) the system.

Considering now the set of components of one given level of the decomposition tree, the relation "uses the service of" — a special form of the relation "interacts with" — enables an accurate definition of a special class of components: if and only if *all* the components of the level may be hierarchically situated through the relation "uses the service", then they are **layers** [44][3]. Stated in equivalent ways, components of a given detail level are layers a) if any two components of that level play a fixed role with respect to this relation, i.e. either consumer or producer, equivalently b) if the graph of the relation is a single branch tree. Conversely, if their respective consumer and producer roles can change, or if their interactions are not governed by such a relation, they are simply components. It is noteworthy that the notion of service has been naturally — and implicitly — generalized with respect to the layers: the service delivered by a given layer is its behavior as perceived by the upper layer, where the term "upper" has to be understood with respect to the relation "uses the service of"[4]. It is noteworthy that the relation "uses the service of" induces an ordering of the time scales of the various layers: time granularity usually does not decrease with increasing layers. Considering the same previous example as in figure 4 leads to the model of figure 5.

Structuring into many layers may be considered for design purposes. Their actual relationship at execution is generally different: compilation removes — partially at least — the structuring, and several layers may, and generally are, executed on a single one. A third type of relation has thus to be considered : *is interpreted by*. The *interpretive interface* [2] between two layers, or sets of layers, is characterized by the provision of objects and operations to manipulate those objects. A system may

---

3 Giving an accurate definition of the notion of layer has been felt necessary due to the current abuse of the term: a layer is *more* than a level of detail. The way a layer has been defined here is consistent with the usage in the domain of protocols, as in the ISO/OSI model.

4 The two hierarchies — levels of details, layers — induced by the relations "is composed of" and "uses the service of" may be considered as *transverse*, just as *transverse orders* in a tree are defined [68].

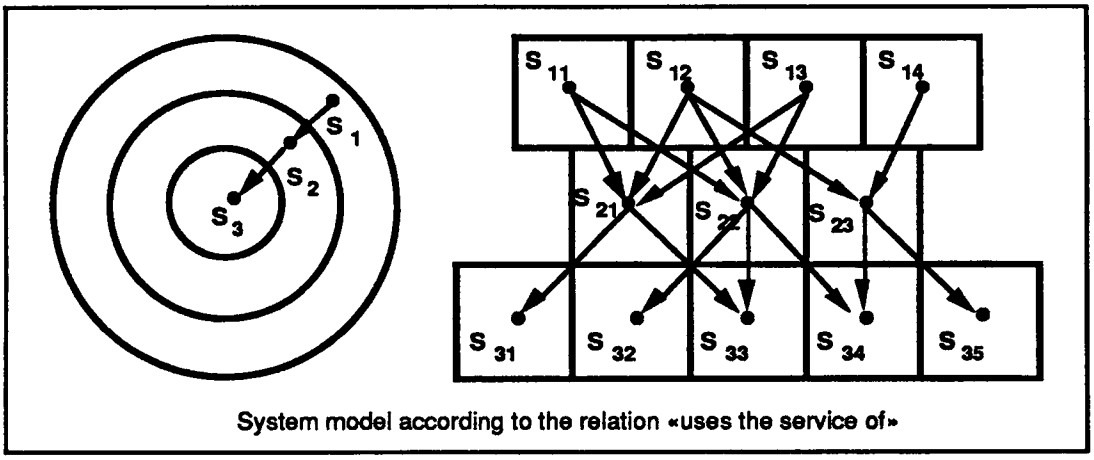

Figure 5 - Layers of a system

then be viewed as a hierarchy of *interpreters*, where a given interpreter may be viewed as providing a "concrete" representation of "abstract" objects in the above interpreter; this concrete representation is itself an abstract object for the interpreter below the considered one.

Considering again the same previous example, hardware layer interprets both the application and the executive software layers. However, the executive software may be viewed as an *extension* [2] of the hardware interpreter, e.g. through a) the provision of "supervisor call" instructions or b) the prevention of invoking certain "privileged" instructions. This leads to the system model depicted by figure 6.

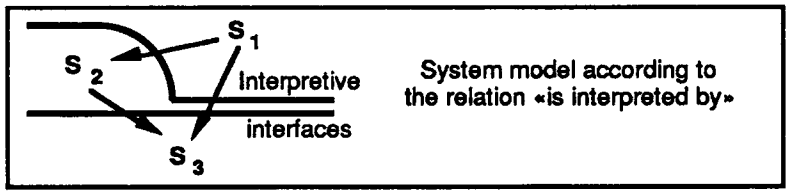

Figure 6 - System interpreters

Finally, it is noteworthy that the expression "abstraction level" has not been used so as to avoid confusion: all the hierarchies which have been defined are strictly speaking abstractions.

2.2.2 Behavior of a single-interpreter system

A system is assumed to be composed of C components, of respective failure rates λ_i, i=1,...,C. The system behavior with respect to the execution process is modeled through a Markov chain with the following parameters:

- S: number of the states of the chain, a state being defined by the components under execution,
- $1/\gamma_j$: mean sojourn time in state j, j=1,...,S.
- q_{jk} = P{ system makes a transition from state j to state k | start or end of execution of one or several components }, j=1,...,S, k=1,...,S, $\sum_{k=1}^{S} q_{jk} = 1$

A system failure is provoked by the failure of any of its components. The system failure rate ξ_j in state j is thus the sum of the failure rates of the components under execution in this state:

$$\xi_j = \sum_{i=1}^{C} \delta_{i,j} \lambda_i \quad , j=1,...,S \tag{2.7}$$

where $\delta_{i,j}$ is equal to 1 if component i is under execution in state j, else it is equal to 0.

The system failure behavior may be modeled by a Markov chain with S+1 states, where the system delivers correct service in the first S states (components are under execution without failure occurrence); state S+1 is the failure state, which is an absorbing state. Let $\mathbb{A} = [\ a_{jk}\ ]$, $j=1,\ldots,S$, $k=1,\ldots,S$, be the transition matrix associated to the non-failed states. This matrix is such that:

- its diagonal terms a_{jj} are equal to $-(\gamma_j+\xi_j)$,
- its non-diagonal terms a_{jk}, $j \neq k$, are equal to $q_{jk}\ \gamma_j$.

The matrix $\mathbb{A}$ may be viewed as the sum of two matrices $\mathbb{A}'$ and $\mathbb{A}''$ such that:

- the diagonal terms of $\mathbb{A}'$ are equal to $-\gamma_j$, its non diagonal terms being equal to $q_{jk}\ \gamma_j$,
- the diagonal terms of $\mathbb{A}''$ are equal to $-\xi_j$, its non diagonal terms being equal to 0.

The system behavior can thus be viewed as resulting from the superimposition of two processes:

- the execution process, of parameters γ_j and q_{jk} (transition matrix $\mathbb{A}'$),
- the failure process, governed by the failure rates ξ_j (transition matrix $\mathbb{A}''$).

A natural assumption is that the failure rates are small with respect to the rates governing the transitions from the execution process, or, equivalently, that a large number of transitions consecutive to the execution process will take place before failure occurrence — a system which would not satisfy this assumption would not be of much practical interest. This assumption is expressed by:

$$\gamma_j \gg \xi_j \tag{2.8}$$

Adopting a Markov approach for modeling the system behavior resulting from the compound execution-failure process is based on this assumption. Similar models have been proposed in the past for software systems [52, 17, 41], with however less generality than here, since those models assumed sequential execution (one component only executed at a time). The Markov approach was justified in those references as follows:

- heuristically in [17, 42], by analogy with performance models in the first reference, and with availability models in the second one,
- from a weaker assumption, semi-Markov, in [52]: it is shown there that the compound process — execution and failure — converges towards a Poisson process, and that the contribution of the distribution functions of the component execution times is limited to their first moments.

By definition, the system failure rate $\lambda(t)$ is given by:

$$\lambda(t) = \lim_{dt \to 0} \frac{1}{dt}\ P\{\text{failure between t and t+dt} \mid \text{no failure between initial instant and t}\ \}$$

Let $Pj(t)$ denote the probability that the system is in state j. It follows that:

$$\lambda(t) = \left(\sum_{j=1}^{S} \xi_j\ Pj(t) \right) \Big/ \left(\sum_{j=1}^{S} Pj(t) \right) \tag{2.9}$$

The consequence of the assumption expressed by relation (2.8) is that the execution process converges towards equilibrium before failure occurrence. The vector $\alpha = [\ \alpha_j\ ]$ of the equilibrium

probabilities is solution of $\alpha . \mathbb{A}' = \mathbb{0}$, with $\sum_{j=1}^{S} \alpha_j = 1$. Thus, $P_j(t)$ converges towards α_j before failure occurrence, and relation (2.9) becomes[5]:

$$\lambda = \sum_{j=1}^{S} \alpha_j \xi_j \tag{2.10}$$

Relation (2.10) may be rewritten as follows, accounting for relation (2.7):

$$\lambda = \sum_{j=1}^{S} \alpha_j \sum_{i=1}^{C} \delta_{i,j} \lambda_i = \sum_{i=1}^{C} \lambda_i \sum_{j=1}^{S} \delta_{i,j} \alpha_j \tag{2.11}$$

Let $\pi_i = \sum_{j=1}^{S} \delta_{i,j} \alpha_j$. Relation (2.10) becomes:

$$\lambda = \sum_{i=1}^{C} \pi_i \lambda_i \tag{2.12}$$

This relation has a simple physical interpretation:

- α_j represents the average proportion of time spent in state j in the absence of failure, thus π_i is the average proportion of time when component i is under execution in the absence of failure; it is noteworthy that the sum of the π_i's can be larger than 1: $0 \leq \sum_{i=1}^{C} \pi_i \leq C$,
- λ_i is the failure rate of component i assuming continuous execution.

The term $\pi_i \lambda_i$ can thus be considered as the *equivalent* failure rate of component i.

Relation (2.12) deserves a few comments. Let us consider first hardware systems. It is generally considered that all components are continuously active. This corresponds to making all the π_i's equal to 1, leading to the usual relation $\lambda = \sum_{i=1}^{C} \lambda_i$.

Let us consider now software systems. The central question is: how to estimate the component failure rates? There are two basic — and opposite — approaches:

- exploiting results of repetitive run experiments without experiencing failures, through statistical testing [22, 72, 55],
- exploiting failure data, using a reliability growth model, the latter being applied to each software component, as performed in [34].

It is important to stress the data representativeness in either approaches; a condition is that the data are collected in a representative environment, i.e. being relative to components in interaction — real or simulated — with the other system components. If this condition is not fulfilled, a distinction has to be made between the interface failure rates (characterizing the failures occurring during interactions with other components) and the internal component failure rates, as in [52], where the expression of a

---

5 Another approach to this result is as follows: a given system component will be executed, thus the transition graph between the non-failed states is strongly connected. As a consequence, the matrix $\mathbb{A}$ is irreducible and it has one real negative eigenvalue whose absolute value σ is lower than the absolute values of the real parts of the other eigenvalues [64]. The system failure behavior is then asymptotically a Poisson process of rate σ. In our case, the asymptotic behavior is relative to the execution process; therefore a) it is reached rapidly, and b) $\sigma = \lambda$, thus the system reliability is: $R(t) = \exp(-\lambda t)$.

component failure rate has the form $\lambda_i = \zeta_i + \gamma_i \sum_j q_{ij}\, \rho_{ij}$, the ζ_i being the internal component failure rates and the ρ_{ij} the interface failure probabilities; this leads to a complexity in the estimation of the order of C^2 instead of C.

An important question is "how to account for different environments?". If this question is interpreted as estimating the reliability of a software system of a base of deployed software systems, then the approach indicated in section 2.1 where the failure rate was considered as a random variable can be extended here, the π_i's being considered as random variables as well. Another interpretation of the above question is: knowing the reliability in a given environment, how to estimate the reliability in another environment. Let us consider the case of sequential software systems. The parameters characterizing the execution process are then defined as follows:

- $1/\gamma_i$: mean execution time of component i, i=1,...,C.
- $q_{ij} = P\{$ component j starts execution | end of execution of component i $\}$, i=1,...,C, j=1,...,C, $\sum_{j=1}^{C} q_{ij} = 1$

The Markov chain modeling the compound execution-failure process is a (C+1) state chain, state i being defined by the execution of component i, and the π_i's reduce to the α_i's (in the case of sequential software, $\delta_{ii} = 1$ and all others are zeros). We have $\lambda_i = p_i\, \gamma_i$, i=1,...,C, where p_i is the failure probability at execution of component i; whence:

$$\lambda = \sum_{i=1}^{C} \pi_i\, \gamma_i\, p_i = \sum_{i=1}^{C} \eta_i\, p_i \qquad (2.13)$$

where $\eta_i = \pi_i\, \gamma_i$ is the visit rate of state i at equilibrium. The η_i's have a simple physical interpretation, as $1/\eta_i$ is the mean recurrence time of state i, i.e. the mean time duration between two executions of component i in the absence of failure. Relation (2.13) is of interest as it enables the distinction to be made between a) the — continuous time — execution process, and b) the — discrete time — failure process conditioned upon execution. If the p_i's are intrinsic to the considered software, and independent of the execution process, then it is possible to infer the software failure rate for a given environment from the knowledge of the η_i's for this environment and the knowledge of the p_i's. The condition for this assumption to be verified in practice is that it is possible to find a suitable decomposition into components: the notion of component for a software is highly arbitrary, and the higher the number of components considered for a given software, the smaller the state space of each component, so the higher the likelihood of providing a satisfactory coverage of the input space for the component. A limit to such an approach is that the higher the number of components, the more difficult is the estimation of the η_i's. It is noteworthy that time granularity (and near decomposability [19]) can offer criteria to find suitable decompositions.

2.2.3 Behavior of a multi-interpreter system

When a system is considered as a hierarchy of interpreters, as defined in § 2.1.1, the execution relative to the selection of an input point for the highest interpreter (which directly interprets the requests originating from the system user) is supported by a sequence of input point selections in the interpreter immediately below, and so on up to the lowest considered interpreter.

Let us assume the system to be composed of I interpreters, the 1st interpreter being the top of the hierarchy, and the Ith interpreter its base.

Each interpreter may be faulty, and each interpreter may be submitted to erroneous inputs. The failure of any interpreter during the computations relative to the input point selection of the interpreter immediately higher will lead to the failure of the latter, and thus by propagation will lead to the failure of the top interpreter, i.e. to system failure. Adopting the terminology of the classical system reliability theory, the interpreters of the hierarchy constitute a «series» system. An intuitive deduction is that the system failure rate is equal to the sum of the failure rates of the interpreters of the hierarchy. If $\lambda_i(t)$, i=1,...,I, denotes the failure rate of interpreter i, we then have:

$$\lambda(t) = \sum_{i=1}^{I} \lambda_i(t) \tag{2.14}$$

A demonstration of this result is given hereafter. Let the following be defined:

- $\nu_i(j_{i-1})$, i=1,...,I, is the number of input points whose execution by interpreter i corresponds to the selection of the (j_{i-1})th input point for interpreter i-1; $\nu_1(j_0)$ is equal to k, the number of selections of input points performed by the user for the considered computation run;
- $p_i(j_i) = P\{$ Failure of interpreter i at selection of the j_ith input point | no failure at the previous input point selections$\}$, $j_i = 1,...,\nu_i(j_{i-1})$, i=1,...,I;
- $t_e(j_i)$, i=1,...,I, is the execution time relative to the selection of the j_ith input point for interpreter i, $j_i = 1,...,\nu_i(j_{i-1})$, i=1,...,I.

Relations (2.1) to (2.5) become:

$$R_d(k) = \prod_{j_1=1}^{k} [\,1 - p_1(j_1)\,] \prod_{j_2=1}^{\nu_2(j_1)} [\,1 - p_2(j_2)\,] \;\ldots\; \prod_{j_I=1}^{\nu_I(j_{I-1})} [\,1 - p_I(j_I)\,]$$

$$t = \sum_{j_1=1}^{k} t_e(j_1) = \sum_{j_1=1}^{k} \sum_{j_2=1}^{\nu_2(j_1)} t_e(j_2) = \sum_{j_1=1}^{k} \sum_{j_2=1}^{\nu_2(j_1)} \cdots \sum_{j_I=1}^{\nu_I(j_{I-1})} t_e(j_I)$$

$$\lambda_i(j_i) = \lim_{\forall j_i\, t_e(j_i)\to 0} \frac{p_i(j_i)}{t_e(j_i)}$$

$$R_d(k) = \prod_{j_1=1}^{k} \{1 - \lambda_1(j_1)\, t_e(j_1) + o[t_e(j_1)]\} \;\ldots\; \prod_{j_I=1}^{\nu_I(j_{I-1})} \{1 - \lambda_I(j_I)\, t_e(j_I) + o[t_e(j_I)]\}$$

$$R(t) = \lim_{\forall j\, \forall j_i\, t_e(j_i)\to 0} R_d(k) = \exp\left(- \sum_{i=1}^{I} \int_0^t \lambda_i(\tau)\, d\tau \right) = \exp\left(- \int_0^t \sum_{i=1}^{I} \lambda_i(\tau)\, d\tau \right)$$

This demonstrates relation (2.14)

Let us consider now that each interpreter is composed of C_i components, i=1,...,I. At execution, a component of interpreter i will use services of one or more components of interpreter i+1, and so on. We may thus define trees of utilisation of services provided by components of interpreter i+1 by the components of interpreter i, as indicated on figure 7.

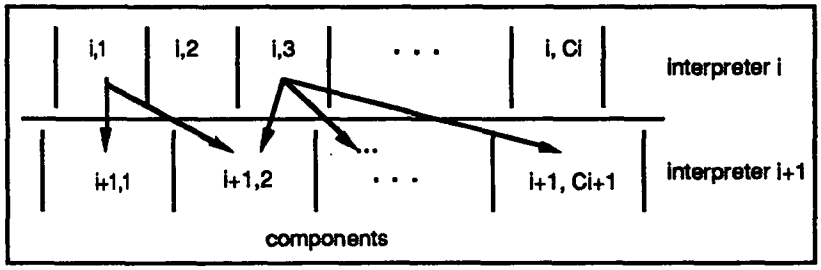

Figure 7 - Utilisation trees between components of interpreters

For each pair of adjacent interpreters, it is thus possible to associate a service utilisation matrix $\mathbb{U}_{i,i+1} = [\ U_{jk}\ ]$, $j=1,\ldots,C_i$, $k=1,..,C_{i+1}$. $\mathbb{U}_{i,i+1}$ is a connectivity matrix whose terms U_{jk} are such that:

- $U_{jk} = 1$ if component j of interpreter i utilises at execution the services of component k of interpreter i+1,
- else $U_{jk} = 0$.

Let us define the following failure rate vectors:

- $\mathbb{\Lambda}_i = [\ \lambda_{i,j}\ ]$, $i=1,\ldots,I$, $j=1,\ldots,C_i$, where $\lambda_{i,j}$ is the failure rate of component j of interpreter i,
- $\mathbb{\Omega}_i = [\ \omega_{i,j}\ ]$, $i=1,\ldots,I$, $j=1,\ldots,C_i$; $\omega_{i,j}$ is the aggregated failure rate of component j of interpreter i; the term «aggregated» means that the failure rates of components of interpreters i+1, ...I needed for execution are accounted for.

The vectors $\mathbb{\Omega}_i$ are solutions of the following matrix equation:

$$\mathbb{\Omega}_i = \mathbb{\Lambda}_i + \mathbb{U}_{i,i+1}\ \mathbb{\Omega}_{i+1}\ ,\ i=1,\ldots,I\text{-}1,\ \mathbb{\Omega}_I = \mathbb{\Lambda}_I$$

It then follows that:

$$\mathbb{\Omega}_1 = \sum_{k=1}^{I} \mathbb{V}_k\ \mathbb{\Lambda}_k$$

$\mathbb{V}_k$ is the accessibility matrix of the top interpreter to interpreter k:

- $\mathbb{V}_1$ is the identity matrix of dimensions $(C_1 x C_1)$,
- $\mathbb{V}_k = \mathbb{U}_{1,2} \otimes \mathbb{U}_{2,3} \ldots \otimes \mathbb{U}_{k-1,k}$, $k=2,\ldots,I$; the symbol $\otimes$ denotes the Boolean product of matrices: a given component can contribute through its failure rate once only.

When applying relation (2.11) to the components of the top interpreter of the hierarchy, we obtain the system failure rate:

$$\lambda = \sum_{j=1}^{C_1} \pi_{1,j}\ \omega_{1,j}$$

where $\pi_{1,j}$ is the proportion of time for which component j of the top interpreter is being executed, with an idle component period being characterized by $\omega_{1,j} = 0$.

Let us consider the specific case, of high practical importance, of a system constituted of two interpreters: a software interpreter and a hardware interpreter. It is assumed that the software components are executed sequentially, and that all hardware components are together involved in the execution; we further assume that the system is in stable operating conditions. In what follows, indices S and H are relative to software and to hardware, respectively. Applying the above approach leads to the following relations:

$$\omega_{S,j} = \lambda_{S,j} + \sum_{k=1}^{C_H} \lambda_{H,k}$$

$$\lambda = \sum_{j=1}^{C_S} \pi_{S,j}\ \omega_{S,j} = \sum_{j=1}^{C_S} \pi_{S,j}\ \lambda_{S,j} + \sum_{k=1}^{C_H} \lambda_{H,k} \qquad (2.15)$$

The intuitive result expressed by relation (2.15) has thus been established through a rigourous approach.

3 Failure behavior of an X-ware system with service restoration

In the previous section, the behavior of atomic and multi-component systems without taking into account the effects of service restoration was characterized, enabling expressions of the failure rate of such systems and of the reliability to be derived. In this section, service restoration is taken into account allowing system behavior resulting from the compound action of the failure and restoration processes to be modeled. Restoration activities may be either a pure restart (supplying the system with an input pattern different from the one which led to failure) or performed after introduction of modifications (corrections only or/and specification changes).

System behavior is first characterized through the evolution of it failure intensity in the first sub-section. The second sub-section introduces the various maintenance policies that can be carried out. The third and fourth sub-sections address respectively reliability and availability modeling.

3.1 Characterization of system behavior

The nature of the operations to be performed in order that the service be restored, i.e. delivered again to its user(s), after a failure has occurred, enables stable reliability or reliability growth to be identified, which may be defined as follows:

- **stable reliability**: the system's ability to deliver proper service is *preserved* (stochastic identity of the successive times to failure);
- **reliability growth**: the system's ability to deliver proper service is *improved* (stochastic increase of the successive times to failure).

Practical interpretations are as follows:

- stable reliability: at a given restoration, the system is identical to what it was at the previous restoration; this corresponds to the following situations:
  - in the case of a hardware failure, the failed part is changed for another one, identical and non failed,
  - in the case of a software failure, the system is restarted with an input pattern different from the one having led to failure;
- reliability growth: the fault whose activation has led to failure is diagnosed as a design fault (in software or in hardware) and is removed.

Reliability *decrease* (stochastic decrease of the successive times to failure) is theoretically, and practically, possible. In such a case, it has to be hoped that the decrease is limited in time, and that reliability is globally growing over a long observation period of time.

Reliability decrease may originate from:

- introduction of new faults during corrective actions, whose probability of activation is greater than for the removed fault(s),
- introduction of a new version, with modified functionalities,
- change in the operating conditions, e.g. an intensive testing period (see [34], where such a situation is depicted),
- dependencies between faults: some software faults can be masked by others, i.e. they cannot be activated as long as the latter are not removed [62]; removal of the masking faults will have as a consequence an increase in the failure intensity.

The reliability of a system is conveniently illustrated by the failure intensity, as it is a measure of the frequency of the system failures as noticed by its user(s). Failure intensity is typically first decreasing (reliability growth), due to the removal of residual design faults, either in the software or in the hardware. It may become stable (stable reliability) after a certain period of operation; the

failures due to internal faults occurring in this period are due to either physical faults, or to design faults which are admitted not to be removed. Failure intensity generally exhibits an increase (reliability decrease) upon the introduction of new versions incorporating modified functionalities; it then tends towards an asymptote again, and so on. It is noteworthy that such a behavior is not restricted to the operational life of a system, but also applies to situations occurring during the development phase of a system, e.g. a) during incremental development [22], or b) during system integration [49, 73].

Typical variations of the failure intensity may be represented as indicated on figure 8, curve (a). Such a curve depends on the granularity of the observations, and may be felt as resulting from the smoothing of more noticeable variations (curve (b)); it may in turn be smoothed into a continuously decreasing curve (c). Although such a representation is very general and covers many practical situations, there are situations which exhibit discontinuities important enough such that the smoothing process cannot be considered as reasonable (e.g. upon introduction of a new system generation).

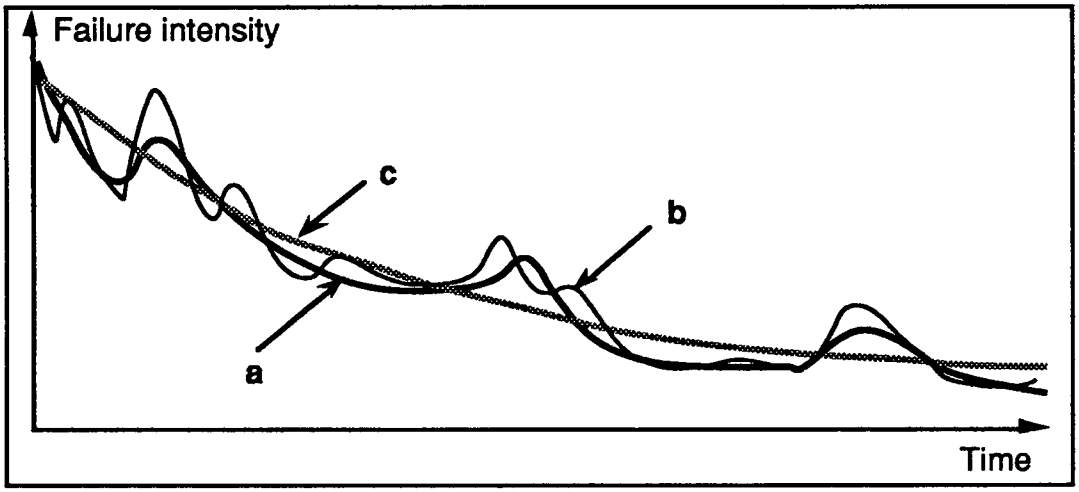

Figure 8 - Typical variations of a system's failure intensity

3.2 *Maintenance policies*

The rate of reliability growth (i. e. failure intensity decrease) is closely related to the correction and maintenance policies adopted for the system [36]. These policies may consist of either a) system modification after each failure, or b) system modification after a given number of failures, or even c) preventive maintenance, i.e. introduction of modifications without any failure observed on the considered system. The status of a system between two modifications will be termed a *version.*

A policy which accepts as special cases the various policies just mentioned is as follows: the j-th system modification takes place after a_j failures have occurred since the (j-1)-th system modification, which means that version j experiences a_j failures.

Concerning the times to failure, let:

- $X_{j,i}$ denote the time between service restoration following the (i-1)-th failure and the i-th failure of version j,
- Z_j denote the time between service restoration following the a_j-th failure of version j and service interruption for introduction of the j-th modification, i.e. for introduction of version (j+1).

Considering now the times to restoration, two types of service restoration are to be distinguished: service restoration due to system restart after failure and service restoration after introduction of a new version.

Let:

- $Y_{j,i}$ denote the restart duration after the i-th failure of version j,
- W_j denote the duration necessary for the introduction of the j-th modification, the modification itself may have been performed off-line.

Let finally T_j denote the time between two version introductions; we have:

$$T_j = \sum_{i=1}^{a_j} (X_{j,i} + Y_{j,i}) + Z_j + W_j \quad , j = 1,2,...$$

The relationship between the various time intervals is given in figure 9.

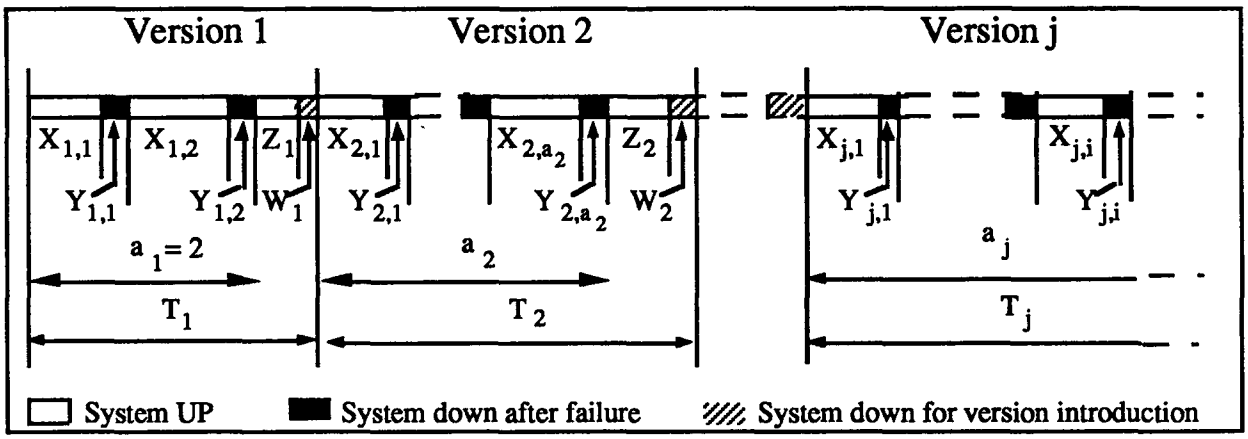

Figure 9 - Relationship between the various time intervals.

The number of failures between two modifications (a_j) characterize the policy for maintenance and service restoration; it depends on several factors such as:

- the failure rate of the system,
- the nature of the faults (e.g., time needed to diagnosis the fault, the consequence of the failure due to the activation of this fault...),
- the considered phase in the life-cycle and may vary for a given system within the same phase,
- the availability of the maintenance team.

Three — extreme — special cases of this general policy are:

a) $a_j = 1$ and $Z_j = 0\ \forall j$: service is restored only after a system modification has been performed; this case relates to:
   - a usual hypothesis for several software reliability (growth) models,
   - the case of critical systems, after a (potentially) dangerous failure occurrence;

b) $a_1 = \infty$ and $Z_j = 0\ \forall j$: service is restored without any system modification ever being performed (stable reliability); this case relates to:
   - hardware, when maintenance consists in replacing a failed part by an identical — new — one,
   - software, when no maintenance is performed, service restoration corresponding always to a restart with an input pattern different from the pattern having led to failure,

c) $a_j = 0$: the (j+1)-th version is introduced before any failure occurrence since the last modification; this case corresponds to preventive maintenance, either corrective, adaptive or perfective.

Although this policy is more general than the usually considered policies (the special cases just above), it is noteworthy that it is however a simplification of real life, as it does not explicitly model the phenomena of a) interweaving of failures and corrections [35], and b) failure re-discoveries [1].

3.3 Reliability modeling

We focus here on the failure process, and thus do not consider the times to restoration, nor the (possible) time interval between a failure and the introduction of a modification, i.e. we assume the $Y_{j,i}$'s, W_j's and Z_j's to be zero which means that the failure instants are also restoration instants.

Let:

- $t_0 = 0$ denote the considered initial instant (the system is assumed non-failed),
- n=1,2,... denote the number of failures: as the n-th system failure is in fact the i-th failure of version j, the relationship between n, i and j is:

$$n = \left(\sum_{k=1}^{j} a_{k-1} \right) + i \ , \ j=1,2,\dots \ , \ i=1,\dots a_j, \ a_0=0 \tag{3.1}$$

- t_n, n=1,2,... denote the instant of failure occurrence,
- $f_{\chi_j}(t)$, j=1, 2,..., denote the probability density functions (pdf) of the times to failure Xj,i, and $sf_{\chi_j}(t)$ denote their survival function (the one's complement of its distribution function): the $X_{j,i}$'s are assumed stochastically identical for a given version,
- $\phi_n(t)$ and $\Phi_n(t)$ denote respectively the pdf and the distribution function of the instants of failure occurrence, n=1,2,... ,
- N(t) denote the number of failures having occurred in [0,t] and H(t) its expectation: H(t) = E[N(t)].

Performing derivations adapted from the renewal theory (see e.g. [25]) is relatively straightforward, provided that the $X_{j,i}$'s are assumed stochastically independent. This assumption, although usual in both hardware and software models, is again a simplification of real life. The T_j's can reasonably be considered as stochastically independent as resuming execution after introduction of a modification generally involves a so-called "cold restart"; it has however to be mentioned that imperfect maintenance, whose consequences were noticed a long time ago [51], is also a source of stochastic dependency. The stochastic independence of the $X_{j,i}$'s for a given j depends on a) the extent to which the internal state of the system has been affected, and on b) the nature of operations undertaken for execution resumption, i.e. whether they involve state cleaning or not.

We get, under the stochastic independence assumption:

$$\phi_n(t) = \left(\circledast_{k=1}^{j-1} f_{\chi_k}(t)^{*a_k} \right) * \left(f_{\chi_j}(t)^{*i} \right), \ j = 1,2,\dots \ , \ i = 1,\dots,a_j, \ a_0=0 \tag{3.2}$$

where the symbol * denotes the convolution operation, $f_{\chi_k}(t)^{*a_k}$ denotes the a_k-fold convolution of $f_{\chi_k}(t)$ by itself, and $\circledast_{k=1}^{j} f_{\chi_k}(t)$ denotes the convolution of $f_{\chi_1}(t),\dots,f_{\chi_j}(t)$. Relation (3.2) is derived from relation (3.1), with the first term covering for j-1 versions and the second term covering the i failures of version j.

We have:

$$P\{ N(t) \geq n \} = P\{ t_n < t \} = \Phi_n(t)$$

$$P\{ N(t) = n \} = P\{ t_n < t < t_{n+1} \} = \Phi_n(t) - \Phi_{n+1}(t)$$

$$H(t) = \sum_{n=1}^{\infty} n \, P\{ N(t) = n \} = \sum_{n=1}^{\infty} n \, [\Phi_n(t) - \Phi_{n+1}(t)]$$

$$H(t) = \sum_{n=1}^{\infty} n\,\Phi_n(t) - \sum_{n=1}^{\infty} (n-1)\,\Phi_n(t) = \sum_{n=1}^{\infty} \Phi_n(t)$$

Let h(t) denote the rate of occurrence of failure [5], or ROCOF, $h(t) = \frac{dH(t)}{dt}$; whence:

$$h(t) = \sum_{n=1}^{\infty} \phi_n(t) = \sum_{j=1}^{\infty} \sum_{i=1}^{a_j} \left(\circledast_{k=1}^{j-1} \left(f_{X_k}(t)^{*a_k} \right) * \left(f_{X_j}(t)^{*i} \right) \right) \qquad (3.3)$$

As we do not consider simultaneous occurrence of failures, the failure process is regular or orderly [5], and the ROCOF is then the *failure intensity* [5].

When considering reliability growth, a usual reliability measure considered is conditional reliability [26, 53, 61]: given that the system has experienced n failures, conditional reliability is the survival function associated with failure n+1. It is defined by:

$$R_{n+1}(\tau) = P\{X_{j,i+1} > \tau \mid t_n\} = sf_{X_j}(\tau), \text{ for } i < a_j$$

$$R_{n+1}(\tau) = P\{X_{j+1,1} > \tau \mid t_n\} = sf_{X_{j+1}}(\tau), \text{ for } i = a_j$$

where n is as defined by relation (3.1).

This measure is mainly of interest when considering a system in its development phase, as one is then concerned by the time to next failure.

However, when considering a system in operational life, the interest is in failure-free time intervals τ whose starting instants are not necessarily conditioned on system failures, that is they can take place at any time t. In this case, the user is thus interested in the reliability over a given time interval independently of the number of failures experienced. The system reliability will then be the probability that the system experiences no failure during the time interval $[t, t+\tau]$.

Let us consider the following exclusive events:

$$E_0 = \{ t+\tau < t_1 \} \qquad E_n = \{ t_n < t < t+\tau < t_{n+1} \},\ n = 1,2, \ldots$$

The event E_n means that exactly n failures occurred prior to instant t, and that no failure occurs during the interval $[t, t+\tau]$. The absence of failure during $[t, t+\tau]$ is the union of all events E_n, $n = 0,1,2,\ldots$ The system reliability is then, owing to the exclusivity of the events E_n:

$$R(t,t+\tau) = \sum_{n=0}^{\infty} P\{E_n\}$$

The probability of event E_n is:

$$P\{E_n\} = P\{ t_n < t < t+\tau < t_n + X_{j,i+1} \} \qquad (3.4)$$

$$P\{E_n\} = \int_0^t P\{x < t_n < x+dx\}\, P\{X_{j,i+1} > t+\tau-x\} = \int_0^t sf_{X_j}(t+\tau-x)\ \phi_n(x)\ dx \qquad (3.5)$$

The reliability has thus the following expression:

$$R(t,t+\tau) = sf_{X_1}(t+\tau) + \sum_{j=1}^{\infty} \sum_{i=0}^{a_j-1} \int_0^t sf_{X_j}(t+\tau-x)\ \phi_{\left(\sum_{k=1}^{j} a_{k-1}\right)+i}(x)\ dx \qquad (3.6)$$

which can be written:

$$R(t,t+\tau) = sf_{X_1}(t+\tau) + \sum_{j=1}^{\infty} \sum_{i=0}^{a_j-1} sf_{X_j}(t+\tau) * \phi_{\left(\sum_{k=1}^{j} a_{k-1}\right)+i}(t) \qquad (3.7)$$

This relation is obviously not of easy utilization. However, it is not difficult to get, for $\tau \ll t$, the following relation from relations (3.3) and (3.7):

$$R(t,t+\tau) = 1 - h(t)\,\tau + o(\tau) \qquad (3.8)$$

Besides its simplicity, relation (3.8) is of high practical importance, as it applies to systems for which the mission time τ is small with respect to the system life time t (e.g. systems on board airplanes).

The above derivations constitute a (simple) generalization of the renewal theory and of the notion of renewal process to non-stationary processes; in the classical theory (stationary processes):

- the $X_{j,i}$'s are stochastically identical, i.e., $f_{X_j}(t) = f_X(t)\ \forall j$ (the case where the first time to failure has a distribution different from the subsequent ones is termed as "modified renewal" process in [20, 11]),
- H(t) and h(t) are respectively the renewal function and the renewal density.

Let us consider the case where the $X_{j,i}$'s are exponentially distributed: $f_{X_j}(t) = \lambda_j \exp(-\lambda_j t)$. The inter-failure occurrence times in such a case constitute a piecewise Poisson process. No assumption is made here as to the sequence of magnitude of the λ_j's. We however assume that the failure process is converging towards a Poisson process after r modifications have taken place. This assumption means that either a) no more modifications are performed or b) if some modifications are still performed, they do not affect significantly the failure behavior of the system. Let $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ be the sequence of these failure rates (figure 10).

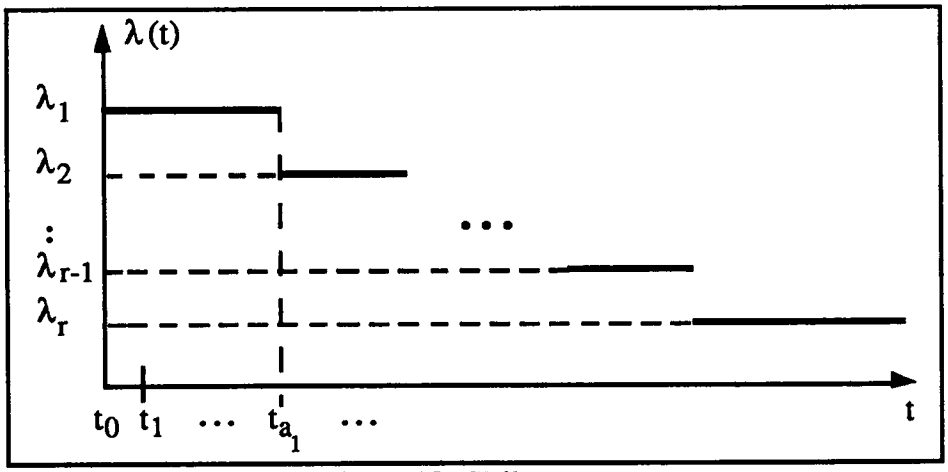

Figure 10 - Failure rate

The Laplace transform $\tilde{h}(s)$ of the failure intensity h(t) (relation (3.3)) is:

$$\tilde{h}(s) = \sum_{j=1}^{r-1} \prod_{k=0}^{j-1} \left(\frac{\lambda_k}{\lambda_k+s}\right)^{a_k} \sum_{i=1}^{a_j} \left(\frac{\lambda_j}{\lambda_j+s}\right)^{i} + \frac{\lambda_r}{s} \prod_{k=1}^{r-1} \left(\frac{\lambda_k}{\lambda_k+s}\right)^{a_k}$$

Derivation of h(t) is very tedious[6]. We shall thus restrict ourselves to summarizing the properties of the failure intensity h(t) which can be derived:

- h(t) is a continuous function of time, with $h(0) = \lambda_1$ and $h(\infty) = \lambda_r$;
- when the a_j's are finite,
  - a condition for h(t) to be a non increasing function of time, i.e. a condition for *reliability growth*, is:
    $$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_j \geq \ldots \geq \lambda_r$$
  - the smaller the a_j's, the faster is the reliability growth,
  - if a (local) increase in the failure rates occurs, then the failure intensity correspondingly (locally) increases,
- when $a_1 = \infty$, no correction takes place and we are in the case of a classical renewal process; then $h(t) = \lambda_1$, which is the formulation of *stable reliability*.

These results are illustrated by figure 11, where the failure intensity is plotted for a_j=a $\forall$j. System reliability $R(t,t+\tau)$ for a given, finite, a (which corresponds to a given curve of figure 11) can then be derived from relation (3.8) for $\tau \ll t$. Figure 12 indicates typical variations of $R(t,t+\tau)$: reliability over mission time τ increases with system life time t.

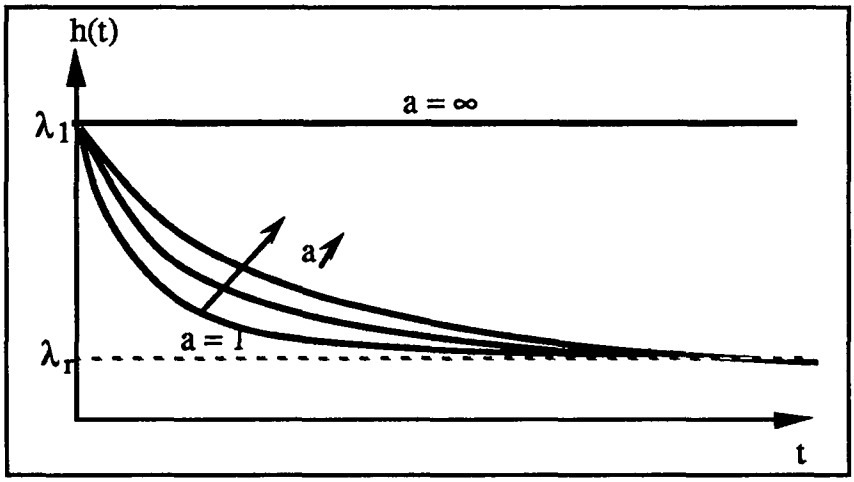

Figure 11 - Failure intensity

The derivations conducted in this section are — although still suffering from some limitations such as the assumed independency between the times to failure necessary for performing the renewal theory derivations — more general than what was previously published in the literature. The resulting model can be termed as a *knowledge model* [48] with respect to the reliability growth models which appeared in the literature, which can be termed as *action models*. Support to this terminology, adapted from the Automatic Control theory, lies in the following remarks:

- the knowledge model allows the various phenomena to be taken into account explicitly and enable a number of properties to be derived; it is however too complex in order to be directly applied in real life for performing predictions,
- the action models, although based on more restrictive assumptions, are simplified models suitable for practical purposes.

---

6 Assuming operation restart after correction only ($a_j = 1 \; \forall \; j$) leads (when going through the steps of the standard procedure involving the decomposition of the Laplace expression into a sum of prime terms, and transforming back the latter into their temporal counterparts) to the following expression of the intensity function:

$$h(t) = \lambda_r + \sum_{i=1}^{r-1} \left(\left(\sum_{k=i}^{r-2} \lambda_i \prod_{j=1 \; j \neq i}^{k} \frac{\lambda_j}{\lambda_j - \lambda_i} \right) + (\lambda_i - \lambda_r) \prod_{j=1 \; j \neq i}^{r-1} \frac{\lambda_j}{\lambda_j - \lambda_i} \right) \exp(-\lambda_i t)$$

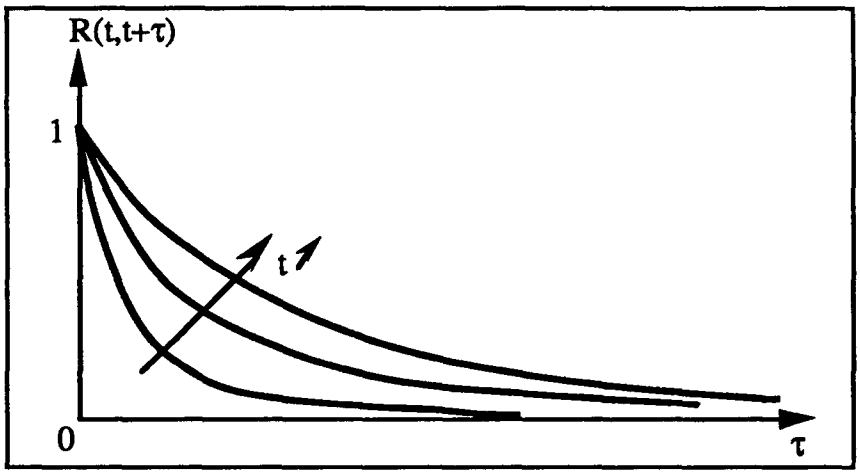

Figure 12 - System reliability

The results obtained from the knowledge model thus derived in this section enable the action models which appeared in the literature to be classified as follows:

1) Models based on the relation between successive failure rates, which can be termed as *failure rate models;* these models describe the behavior between two failures. Two categories of failure rate models can be distinguished according to the nature of the relationship between the successive failure rates:
   - deterministic relationship, which is the case for most failure rate models; see e.g. [33, 69, 60],
   - stochastic relationship [39, 54]; the corresponding models are termed as doubly stochastic reliability growth models in [59].
2) Models based on the failure intensity, thus called *failure intensity models*; these models describe the failure process, and are usually expressed as non-homogeneous Poisson processes; see e.g. [21, 26, 76, 61, 47].

Most of the reliability growth models consider reliability growth stricto sensu, without consideration of possible stable reliability or reliability decrease periods: they assume that the failure rate and/or the failure intensity decrease monotically and become asymptotically zero with time. It is however noteworthy that:

- the so-called S-shaped models [76, 73] relate to initial reliability decrease followed by reliability growth,
- our hyperexponential model [41, 34, 47] relates to reliability growth converging towards stable reliability.

Finally, it is noteworthy that the models we have referenced above have been established specifically for software; however, there is no impairment to apply them to hardware [53]; reciprocally, the Duane's model [23], derived for hardware, has been applied successfully to software [39].

A question of prime importance arises when considering the practical use of the above models: how can they be applied to real systems? Failure data can be collected under two forms: a) times to failure, or b) number of failures per unit of time. Failure rate models are more naturally suited to data in the form of times to failure, whereas failure intensity models are more naturally suited to data in the form of number of failures per unit of time; however, some models are of enough mathematical tractability so as to accommodate both forms of failure data, such as the logarithmic Poisson model [61], or our hyperexponential model [48]. The collection of data under the form "failures per unit of time" is less constraining than the other form, since one does not have to record all failure times; the definition of the unit of time can be varied throughout the life cycle of the system, according to the

amount of failures experienced, e.g. days to weeks during development, weeks to months during operational life.

As the action models are based on precise hypotheses (especially concerning the reliability trends they are able to accommodate, as discussed above), it is helpful to process failure data before model application, in order a) to determine the reliability trend exhibited by the data, and b) to select the model(s) whose assumptions are in agreement with the evidenced trend [37]. Trend tests [5] are a quite natural approach for such a pre-processing. Trend tests are in addition extremely helpful in assisting the management of software development and validation [38, 49], as the relevance of the utilization of reliability growth models during early stages of development and validation is most often questionable: when the observed times to failure are of the order of magnitude of minutes or hours, the predictions performed from such data can hardly predict mean times to failure different from minutes or hours ... which is distant from any expected reasonable reliability for most applications. In addition, when a program under validation becomes reliable enough, the times to failure may simply be large enough in order to make the application of reliability growth models impractical, due to the (hoped for) scarcity of failure data exhibited by one or a few copies of a software system under validation. On the other hand, the utilization of reliability growth models for large bases of deployed software systems is much more convincing (see e.g. [1, 34] for examples of such studies).

3.4 Availability modeling

All the time intervals defined in section 3.2 are now considered, i.e. the times to failure and the times to restoration: the $Y_{j,i}$'s, Z_j's and W_j's are not any longer assumed to be zero. For the sake of simplicity, we shall assume that the times to restoration after failure are stochastically identical for a given version, i.e. $Y_{j,i} = Y_j$.

Let:

- $t''_0 = 0$ denote the considered initial instant (the system is assumed non-failed),
- n denote the number of service restorations that took place before instant t,
- t'_n and t''_n, n=1,2,... denote respectively the instants when correct service is not any longer delivered and when service is restored, either:
  - upon (resp. after) failure, with $n = \sum_{k=1}^{j} (a_{k-1}+1) + i$, j=1,2,... , i=1,...a_j, $a_0=0$
  - upon (resp. after) stopping system operation for modification introduction, with $n = \sum_{k=1}^{j+1} (a_{k-1}+1)$, j=1,2,... , $a_0=0$
- $f_{Y_j}(t)$, $f_{Z_j}(t)$, $f_{W_j}(t)$, j=1, 2,..., denote the probability density functions (pdf) of the Y_j's, Z_j's and W_j's, respectively, and $sf_{Z_j}(t)$ denote the survival function of the Z_j's,
- $\psi_n(t)$ denotes the pdf of the instants of service restoration, n=1,2,...

The derivation of availability is conducted in a way similar to the derivation of reliability (relations (3.2), (3.4) to (3.8)), with the pdf of the instants of service restoration $\psi_n(t)$ replacing the the pdf of the instants of failure occurrence $\phi_n(t)$. We assume as in section 3.3 that the various time intervals under consideration are stochastically independent, which leads to:

- for $n = \sum_{k=1}^{j} (a_{k-1}+1) + i$,

$$\psi_n(t) = \left(\circledast_{k=1}^{j-1} \left(f_{\chi_k}(t) * f_{\gamma_k}(t) \right)^{*a_k} * \left(f_{Z_k}(t) * f_{w_k}(t) \right) \right) * \left(f_{\chi_j}(t) * f_{\gamma_j}(t) \right)^{*i}$$

- for $n = \sum_{k=1}^{j+1} (a_{k-1}+1)$

$$\psi_n(t) = \circledast_{k=1}^{j} \left(f_{\chi_k}(t) * f_{\gamma_k}(t) \right)^{*a_k} * \left(f_{Z_k}(t) * f_{w_k}(t) \right)$$

Let us consider the event $E_n = \{ t''_n < t < t'_{n+1} \}$, n=0,1,2,... The event E_n means that exactly n service restorations took place before instant t, and that the system is non-failed at instant t. The pointwise availability A(t), denoted simply by availability in the following, is then, due to the exclusivity of events E_n:

$$A(t) = \sum_{n=0}^{\infty} P\{E_n\}$$

The probability of event E_n is:

- for $n = \sum_{k=1}^{j} (a_{k-1}+1) + i$:

$$P\{E_n\} = P\left\{ t''_n < t < t+\tau < t''_n + X_{j,i+1} \right\}$$

$$P\{E_n\} = \int_0^t P\{x < t''_n < x+dx\}\, P\{X_{j,i+1} > t-x\} = \int_0^t sf_{\chi_j}(t-x)\ \psi_n(x)\ dx$$

- for $n = \sum_{k=1}^{j+1} (a_{k-1}+1)$:

$$P\{E_n\} = P\left\{ t''_n < t < t+\tau < t''_n + Z_j \right\} = \int_0^t sf_{Z_j}(t-x)\ \psi_n(x)\ dx$$

The expression of A(t) is then:

$$A(t) = sf_{\chi_1}(t) + \sum_{j=1}^{\infty} \left(sf_{\chi_j}(t) * \sum_{i=0}^{a_j-1} \psi_{\left(\sum_{k=1}^{j} (a_{k-1}+1)+i \right)}(t) + sf_{Z_j}(t) * \psi_{\left(\sum_{k=1}^{j+1} (a_{k-1}+1) \right)}(t) \right)$$

Statistical estimation of the availability for a set of systems is given by the ratio of non-failed systems at time t to the total number of systems in the set. When the field data are relative to times to failure and to times to restoration, considering average availability instead of availability eases the estimation process, as average availability is the expected proportion of time a system is non-failed (see e.g. [8]). The average availability over [0,t] is defined by: $A_{av}(t) = \frac{1}{t} \int_0^t A(\tau)\, d\tau$. Denoting

respectively by UT_i the observed times where the system is operational, a statistical estimator of $A_{av}(t)$ is: $\hat{A}_{av}(t) = \frac{1}{t}\sum_{i=1}^{n} UT_i$.

Up to now, we have made no assumption concerning the pdfs of the various times considered. In order to derive properties of the availability, let us consider the case where:

- a modification takes place after each failure, i.e. $a_j = 1, Z_j = W_j = 0 \; \forall \, j$,
- the $X_{j,i}$'s and the Y_j's are exponentially distributed: $f_{X_j}(t) = \lambda_j \exp(-\lambda_j t)$, $f_{Y_j}(t) = \mu_j \exp(-\mu_j t)$, and that both corresponding piecewise Poisson processes converge towards Poisson processes after r modifications have taken place.

The Laplace transform $\tilde{A}(s)$ of the availability is then:

$$\tilde{A}(s) = \sum_{j=1}^{r-1} \left(\frac{1}{\lambda_j+s}\right)\prod_{k=1}^{j-1} \left(\frac{\lambda_k}{\lambda_k+s}\right)\left(\frac{\mu_k}{\mu_k+s}\right) + \frac{1}{s} \; \frac{\mu_r \lambda_r}{\lambda_r+\mu_r+s} \left(\frac{1}{\lambda_r+s}\right)\prod_{k=1}^{r-1} \left(\frac{\mu_k}{\mu_k+s}\right)\left(\frac{\lambda_k}{\lambda_k+s}\right)$$

The times to failure are large with respect to the times to restoration, i.e. $\lambda_j/\mu_j \ll 1$. Performing an asymptotic development with respect to λ_j/μ_j for the unavailability $\overline{A}(t) = 1 - A(t)$ leads to:

$$\overline{A}(t) = \frac{\lambda_r}{\mu_r} + \sum_{j=1}^{r-1} \alpha_j \exp(-\lambda_j t) - \frac{\lambda_1}{\mu_1} \exp(-\mu_1 t) \qquad (3.9)$$

with: $$\alpha_j = \sum_{k=j}^{r-1} \frac{\lambda_k}{\mu_k} \; \frac{\prod_{i=1}^{k-1} \lambda_i}{\prod_{i=1, i\neq j}^{k} (\lambda_i - \lambda_j)} - \frac{\lambda_r}{\mu_r} \prod_{k=1, k\neq j}^{r-1} \frac{\lambda_k}{\lambda_k - \lambda_j}$$

The α_j are related by the equation: $\sum_{j=1}^{r-1} \alpha_j = \frac{\lambda_1}{\mu_1} - \frac{\lambda_r}{\mu_r}$

The following properties can be derived from relation (3.9), which confirm and generalize what had previously been established in [18, 42] through modeling via multi-state Markov and semi-Markov chains:

P1) When reliability becomes stable, unavailability becomes constant: $\overline{A}(\infty) \approx \lambda_r / \mu_r$.

P2) If $(\lambda_1 / \mu_1) \geq (\lambda_r / \mu_r)$ then there is an overshoot of unavailability in comparison with the asymptotic value (λ_r / μ_r).

P3) There is a single unavailability maximum (availability minimum), and if this growth is not nullified by possible increase in the restoration rates, that is if $(\lambda_{j+1} / \mu_{j+1}) \leq (\lambda_j / \mu_j)$, j=1, ...; in the converse, local maxima (minima) occur.

P4) If the piecewise Poisson process of the inter-failure occurrence times is continuously non increasing from λ_1 to λ_r, and if the times to failure are large with respect to the times to restoration, then $\overline{A}_{max} \approx (\lambda_1 / \mu_1)$.

P5) The time to reach the maximum unavailability (minimum availability) is of the order of magnitude of the mean time to restoration $(1 / \mu_1)$.

P6) The changes in availability are significantly more influenced by the stochastic changes in the times to failure than by the stochastic changes in the times to restoration, which can thus be assumed as stochastically identical over the system's life.

Figure 13 gives the typical shape of system unavailability in the presence of reliability growth.

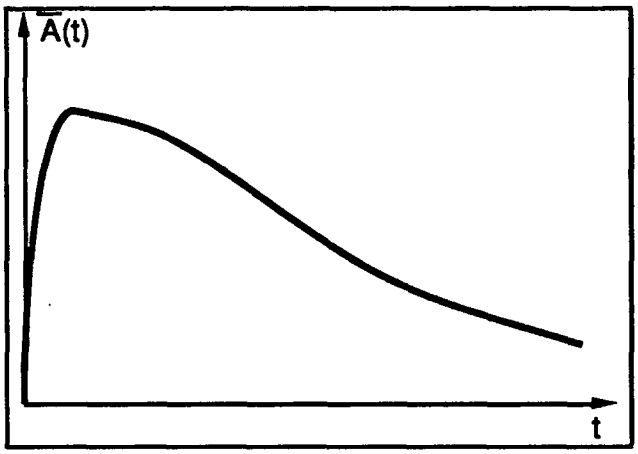

Figure 13 - Typical system unavailability

Although availability of operational computing systems is usually significantly influenced in the field by reliability growth (for instance, the data displayed in [75 for AT&T's ESS-4 show that unavailability is decreased by a factor of 250 during 3.5 years of operation), the only published *action* model for availability in the presence of reliability growth is our hyperexponential model [47, 48].

4 Conclusion

In the second section we have dealt with system behavior up to (next) failure, and in the third section with the sequence of failures when considering restoration actions consecutive to various forms of maintenance. The assumptions we have made for performing the derivations have been carefully stated and commented, and we have widely commented the obtained results in order to situate them with respect to the existing body of results.

The second section can be seen as a generalization of the classical, hardware-oriented, reliability theory in order to incorporate software as well. The third section is in fact concerned with reliability growth phenomena; as most of the published material dealing with reliability growth is software-oriented, this section can be seen as a generalization in order to incorporate hardware as well.

Continuing the utilization of data and models which are application-independent and ignoring design faults in hardware — stable reliability as usually assumed is in fact a very special case — can only lead to a decrease in the confidence attached to the evaluations performed, a consequence being a diminished role of these evaluations. Restricting software reliability to *pursuing* a set of experimental data collected on specific products will certainly not improve its practical utilization.

We have shown in this paper that it is theoretically possible to perform reliability and availability evaluations of *systems*, accounting for hardware *and* software, with respect to the various classes of faults which may affect them. Performing such evaluations in practice is thus mainly a matter of performing experiments and measurements, whose two types appear to be necessary:

- collection of data on several systems produced by a given designer/builder in order to incorporate the influence of its specific characteristics (organization, methods used, etc.); an output of this step, which is in fact concerned with *process modeling*, is the production of models generic to large classes of applications,

- calibration of these previous models for specific systems and specific users through exercising the system with input sequences representative of its future usage, that is applying statistical testing to dependability evaluation [22, 72]; this step is thus concerned with *product modeling*.

It is noteworthy that any failure data handbook comparable to the MIL-HDBK-217 for hardware is certainly currently out of reach for software, and the very concept of such a handbook is likely to be meaningless for software; however, a given designer/builder can certainly gather and exploit data which are significant with respect to its own software products and its customers, i.e. accounting for the influence of its own characteristics. After all, software is an intellectual construction, and as such is highly — if not totally — influenced by the methods and practices of its production process.

What has been presented in this paper clearly indicates a challenge that the theoreticians and practicians of dependability evaluation are faced with: dealing with design faults, and thus achieving the integration of dependability growth phenomena into predictive evaluations. Naturally, the work presented in this paper has to be continued and extended, especially with regard to a) multi-environment situations, and b) fault-tolerant systems.

Acknowledgement

We wish to thank Alain Costes and Mohamed Kaâniche for their constructive suggestions and insightful comments when reading the first version of this paper, as well as Pierre-Jacques Courtois (Philips Research Laboratory in Brussels), Bev Littlewood and Peter Mellor (City University, London), for their helpful comments. The anonymous reviewers greatly helped in improving the final version.

References

1 E.N. Adams, "Optimizing preventive service of software products", *IBM J. of Research and Development*, vol. 28, no. 1, Jan. 1984, pp. 2-14.

2 T. Anderson, P.A. Lee, *Fault Tolerance — Principles and Practice*, Prentice Hall, 1981.

3 J. Arlat, K. Kanoun, J.C. Laprie, "Dependability evaluation of software fault-tolerance", *Proc. 18th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-18)*, Tokyo, June 1988, pp. 142-147, also in *IEEE Trans. on Computers*, vol. 39,, no. 4, April 1990, pp. 504-513.

4 T.F. Arnold, "The concept of coverage and its effect on the reliability model of repairable systems", *IEEE Trans. on Computers*, vol. C-22, June 1973, pp. 251-254.

5 H. Ascher, H. Feingold, *Repairable Systems Reliability: Modeling, Inference, Misconceptions and Their Causes*, Lecture notes in statistics, Vol. 7, 1984.

6 R.L. Aveyard, F.T. Man, "A study on the reliability of the circuit maintenance system 1-B", *Bell System Technical Journal*, vol. 59, Oct. 1980, pp. 1317-1332.

7 A. Avizienis, J.C. Laprie, "Dependable computing: from concepts to design diversity", *Proceedings of the IEEE*, vol. 74, no. 5, May 1986, pp. 629-638.

8 R.E. Barlow, F. Proschan, *Statistical Theory of Reliability and Life Testing*, New York: Holt, 1975.

9 H.A. Bauer, L.M. Croxall, E.A. Davis, "The 5ESS switching system: system test, first-office application, and early field experience", *AT&T Technical Journal*, vol. 64, no. 6, pp. 1503-1522.

10 B. Beyaert, G. Florin, P. Lonc, S. Natkin, "Evaluation of computer systems dependability using stochastic Petri nets", *Proc. 11th IEEE Int. Symp. Fault-Tolerant Computing (FTCS-11)*, Portland, Maine, June 1981, pp. 79-81.

11 A. Birolini, "Some applications of regenerative stochastic processes to reliability theory — Part one: tutorial introduction", *IEEE Trans. on Reliability*, vol. R-23, no. 3, Aug. 1974, pp. 186-194.

12 A. Bobbio, K.S. Trivedi, "An aggregation technique for the transient analysis of stiff Markov chains", *IEEE Trans. on Computers*, vol. C-35, Sept. 1986, pp. 803-814.

13 W.G. Bouricius, W.C. Carter, P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems", *Proc. 24th ACM National Conf.*, 1969, pp. 295-309.

14 W.C. Carter, D.C. Jessep, W.G. Bourricius, A.B. Wadia, C.E. McCarthy, F.G. Milligan, "Design techniques for modular architectures for reliable computer systems", IBM T.J. Watson Report no. 70.208.0002, 1970.

15 X. Castillo, D.P. Siewiorek, "Workload, performance, and reliability of digital computing systems", *Proc. 11th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-11)*, Portland, Maine, June 1981, pp. 84-89.

16 R.C. Cheung, "A user-oriented software reliability model", *IEEE Trans. on Software Engineering*, vol. SE-6, March 1980, pp. 118-125.

17 G.F. Clement, P.K. Giloth, "Evolution of fault tolerant switching systems in AT&T", in *The evolution of Fault-Tolerant Computing*, A. Avizienis, H. Kopetz, J.C. Laprie, Eds, Wien: Springer-Verlag, 1987, pp. 37-54.

18 A. Costes, C. Landrault, J.C. Laprie, "Reliability and availability models for maintained systems featuring hardware failures and design faults", *IEEE Trans. on Computers*, vol. C-27, June 1978, pp. 548-560.

19 P.J. Courtois, *Decomposability: Queing and Computer System Application*, New York: Academic 1977.

20 D.R. Cox, *Renewal Theory*, Methuen' Monographs on Applied Probabibity and Statistics 1962.

21 L.H. Crow, "Confidence interval procedures for reliability growth analysis", Tech. report 1977, US Army Material Syst. Anal. Activity Aberdeen, MD, 1977.

22 P.A. Currit, M. Dyer, H.D. Mills, "Certifying the reliability of software", *IEEE Trans.on Software Engineering*, vol. SE-12, no. 1, Jan. 1986, pp. 3-11.

23 J.T. Duane, "Learning curve approach to reliability monitoring", *IEEE Trans. on Aerospace*, vol. 2, 1964, pp. 563-566.

24 European Space Agency, "Software reliability modelling study", Invitation to tender AO/1-2039/87/NL/IW, Feb. 1988.

25 B.V. Gnedenko, Y.K. Belyayev, A.D. Solovyev, *Mathematical methods of reliability theory*, New York: Academic Press, 1969.

26 A.L. Goel, K. Okumoto : "Time-dependent Error-detection Rate Model for Software and other Performance Measures", *IEEE Trans. on Reliability*, vol. R-28, n°3, Aug. 1979, pp. 206-211.

27 J.N. Gray, "Why do computers stop and what can be done about it?", *Proc. 5th Symp. on Reliability in Distributed Software and Database Systems*, Los Angeles, Jan. 1986, pp. 3-12.

28 A. Grnarov, J. Arlat, A. Avizienis, "On the performance of software fault tolerance strategies", *Proc. 10th IEEE Int. Symp. Fault-Tolerant Computing (FTCS-10)*, Kyoto, Oct. 1980, pp. 251-253.

29 D. Gross, D.R. Miller, "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes", *Operations Research*, vol. 32, no. 2, 1984, pp.343-361.

30 H. Hecht, "Fault-tolerant software", *IEEE Trans. on Reliability*, vol. R-28, Aug. 1979, pp. 227-232.

31 H. Hecht, E. Fiorentino, "Reliability assessment of spacecraft electronics", *Proc. 1987 Annual Reliability and Maintainability Symp.*

32 R.K. Iyer, S.E. Butner, E.J. McCluskey, "A statistical failure/load relationship: results of a multi-computer study", *IEEE Trans. on Computers*, vol. C-31, July 1982, pp. 697-706.

33 Z. Jelinski, P.B. Moranda : "Software Reliability Research", *Proc. Statistical Methods for the Evaluation of Computer System Performance*, Academic Press, 1972, pp. 465-484.

34 K. Kanoun, T. Sabourin, "Software dependability of a telephone switching system", *Proc. 17th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-17)*, Pittsburgh, June 1987, pp. 236-241.

35 K. Kanoun, J.C. Laprie, T. Sabourin, "A method for software reliability growth analysis and assessment", *Proc. 1st Int. Workshop on Software Engineering and its applications,* Toulouse, Dec. 1988, pp. 859-878.

36 K. Kanoun, "Software dependability growth — Characterization, modeling, evaluation", Doctor ès-Sciences thesis, Toulouse Polytechnic National Institute, Sept. 89, LAAS Report no. 89.320; in French.

37 K. Kanoun, M. Bastos Martini, J. Moreira De Souza, "A Method for Software reliability analysis and Prediction — Application to The TROPICO-R Switching System", *IEEE Trans. on Software Engineering,* vol. 17, no. 4, April 1991, pp. 334-344.

38 K. Kanoun, J.C. Laprie, "The role of trend analysis in software development and validation", *Proc. IFAC Int. Conf. on Safety, Security and Reliability (SAFECOMP'91),* 30 Oct.-1 Nov. 1991, Trondheim, Norway.

39 P.A. Keiller, B. Littlewood, D.R. Miller, A. Sofer, "Comparison of software reliability predictions", *Proc. 13th IEEE Int. Symp. Fault-Tolerant Computing (FTCS-13),* Milano, June 1983, pp. 128-134.

40 B.A. Kozlov, U.A. Ushakov, *Reliability Handbook,* Edited by L.H. Koopmans and J. Rosenblat, Holt Rinehart and Winston, Inc.

41 J.C. Laprie, "Dependability modeling and evaluation of hardware-and-software systems", *Proc. 2nd GI/NTG/GMR Conf. on Fault Tolerant Computing*, Bonn, Germany, Sept. 1984, pp. 202-215.

42 J.C. Laprie, "Dependability evaluation of software systems in operation", *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, Nov. 1984, pp. 701-714.

43 J.C. Laprie, "Dependable computing and fault tolerance: basic concepts and terminology", *Proc. 15th Int. IEEE Symp. on Fault Tolerant Computing (FTCS-15)*, Ann Arbor, Michigan, June 1985, pp. 2-11.

44 J.C. Laprie, "Towards an X-ware reliability theory", *Technique et Science Informatiques,* vol. 7, no. 3, 1987, pp. 315-330; in French. Available in English as LAAS report no. 86.376, Dec. 1986.

45 J.C. Laprie, "Harware-and-software dependability evaluation", *Proc. IFIP 11th World Congress,* San Francisco, USA, Aug. 1989, pp. 109-114.

46 J.C. Laprie, "Dependability: a unifying concept for reliable computing and fault tolerance", in *Dependability of Resilient Computers,* T. Anderson Ed., London: Blackwell, 1989, pp. 1-28.

47 J.C. Laprie, C. Beounes, M. Kaâniche, K. Kanoun, "The transformation approach to modeling and evaluation of the reliability and availability growth of systems", *Proc. 20th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-20)*, Newcastle, UK, June 1990, pp.364-371.

48 J.C. Laprie, K. Kanoun, C. Beounes, M. Kaâniche, "The KAT (Knowledge-Action-Transformation) Approach to the Modeling and Evaluation of Reliability and Availability Growth", *IEEE Transactions on Software Engineering*, vol. 17, no. 4, April 1991, pp. 370-382

49 Y. Levendel, "Defects and reliability analysis of large software systems: field experience", *Proc. 19th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-19),* Chicago, June 1989, pp. 238-244.

50 Y. Levendel, "Software Quality Improvement Process: When to Stop Testing", Proc. of *Software Engineering & its Applications*, Toulouse, France, Dec. 1991, pp. 729-749.

51 P.A. Lewis, "A branching Poisson process model for the analysis of computer failure patterns", *J. R. Statist. Soc. B*, vol. 26, no. 3, 1964, pp. 398-456.

52 B. Littlewood, "Software reliability model for modular program structure", *IEEE Trans. on Reliability*, vol. R-30, Oct. 1981, pp. 313-320.

53 B. Littlewood, "Stochastic reliability growth: a model for fault-removal in computer programs and hardware designs", *IEEE Trans. on Reliability,* vol. R-30, no. 4, Oct. 1981, pp. 313-320.

54 B. Littlewood, "Forecasting software reliability", in *Software Reliability Modeling and Identification,* S. Bittanti Ed., Berlin: Springer-Verlag, 1988, pp. 140-209.

55 B. Littlewood, "Limits to evaluation of software dependability", in *Software Reliability and Metrics* (Editors B. Littlewood and N. Fenton) Elsevier, 1991.

56 J.F. Meyer, "On evaluating the performability of degradable computing systems", *Proc. 8th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-8)*, Toulouse, June 1978, pp. 44-49.

57 J.F. Meyer, L. Wei, "Analysis of workload influence on dependability", *Proc. 18th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-18)*, Tokyo, June 1988, pp. 84-89.

58 S.R. McConnell, D.P. Siewiorek, M.M. Tsao, "The measurement and analysis of transient errors in digital computer systems", *Proc. 9th IEEE Int. Symp. on Fault Tolerant Computing (FTCS-9)*, Madison, Wisconsin, June 1979, pp. 67-70.

59 D.R. Miller, "Exponential order statistic models of software reliability growth", *IEEE Trans. on Software Engineering*, vol. SE-12, no. 1, Jan. 1986, pp. 12-24.

60 J.D. Musa : "A theory of Software Reliability and its Application", *IEEE Trans. on Soft. Eng.*, vol. SE-1, Sept. 1975, pp. 312-327.

61 J.D. Musa, K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement", *Proc. Compsac'84*, Chicago, 1984, pp. 230-238.

62 M. Ohba, "Software reliability analysis models", *IBM J. of Research and Development*, vol. 21, no. 4, July 1984, pp. 428-443.

63 P.M. Nagel, J.A. Skrivan, "Software reliability: repetitive run experimentation and modeling", Report NASA CR-165836, Feb. 1982.

64 A. Pages, M. Gondran, *System Reliability*, Eyrolles, Paris, 1980; in French.

65 D.L. Parnas, "On a 'buzzword': hierarchical structure", in *Proc. 1974 IFIP Congress*, pp. 336-339.

66 P.I. Pignal, "An analysis of hardware and software availability exemplified on the IBM 3725 communication controler", *IBM J. of Research and Development*, vol. 32, no. 2, March 1988, pp. 268-278.

67 W.B. Rohn, T.F. Arnold, "Design for low expected downtime control systems", *Proc. 4th Int. Conf. on Computer Communications*, Philadelphia, PA, June 1972, pp. 16-25.

68 B. Roy, *Modern Algebra and Graph Theory*, Paris: Dunod, 1969; in French.

69 M. Shooman : "Operating testing and software reliability during program development", *IEEE Symp. Comput. Software Rel.*, New York, 30 April - 2 May, 1973, pp. 51-57.

70 R.M. Smith, K.S. Trivedi, A.V. Ramesh, "Performability analysis: measures, an algorithm, and a case study, *IEEE Trans. on Computers*, vol. 37, no. 4, April 1988, pp. 406-417.

71 G.E. Stark, "Dependability evaluation of integrated hardware / software systems", *IEEE Trans. on Reliability*, vol. R-36, no. 4, 1987, pp. 440-444.

72 P. Thévenod-Fosse, "Software validation by means of statistical testing: retrospect and future direction", *Dependable Computing and Fault-tolerant Systems, Vol. 4*, (Editors A. Avizienis and J.C. Laprie), Springer-Verlag Wien New York, pp. 25-48.

73 Y. Tohma, K. Tokunaga, S. Nagase, Y. Murata, "Structural approach to the estimation of the number of residual faults based on the hyper-geometric distribution", *IEEE Trans. on Software Engineering*, vol. SE-15, no. 3, March 1989, pp. 345-355.

74 W.N. Toy, "Modular redundancy concept, problems and solutions", EPRI Seminar: Digital Control and Fault-Tolerant Computer Technology, Scottsdale, Arizona, April 1985.

75 J.J. Wallace, W.W. Barnes, "Designing for ultrahigh availability: the Unix RTR operating system", *Computer*, Aug. 1984, pp. 31-39.

76 S. Yamada, S. Osaki, "Reliability growth modeling for software error detection", *IEEE Trans. on Reliability*, vol. R-32, no. 5, 1983, pp. 475-478.

77 S. Yamada, S. Osaki, "Software reliability growth modeling: models and assumptions", *IEEE Trans. on Software Engineering*, vol. SE-11, no. 12, Dec. 1985, pp. 1431-1437.