Dependability Evaluation: a Survey

Karama Kanoun

Invited paper - 2nd IARP/IEEE-RAS Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, Toulouse (France), Octobre 7-8 2002, pp.209-216

Dependability Evaluation: a Survey

Karama Kanoun
LAAS-CNRS, 7 Avenue du Colonel Roche — 31077 Toulouse Cedex 4 — France
Karama. Kanoun @laas.fr

Introduction

Dependability evaluation of computer systems is intended to estimate the presence, creation and consequences of faults on dependability and to facilitate whenever possible, their removal or tolerance. It is usually implemented by means of ordinal or probabilistic evaluation techniques.

Ordinal evaluation consists in analyzing the link between faults (i.e., failure causes) and their effects on system behavior. Its aim is to identify possible system design weaknesses.

Probabilistic evaluation of dependability encompasses both model-based and measurement-based techniques. Model-based evaluation includes both analytical and simulation models. The difference between the two approaches concerns mainly: i) the abstraction level usually considered for describing (modeling) the behavior of the system and ii) the assumptions attached to the distributions of the stochastic processes governing the parameters of the model. In this paper we will primarily refer to analytical modeling. Measurement encompasses both the observation of a real-life system in operation (termed as field measurement here) and controlled experimentation where faults are deliberately injected into the target system so as to accelerate the characterization of its faulty behavior, usually referred to as fault injection experiments.

Numerous proven methods and techniques have been successfully developed and used. These techniques are complementary and are well-suited for various life-cycle phases of computer systems:

- Ordinal evaluation techniques constitute a powerful tool for qualitative system analysis. They are recommended for the early design phases to minimize the cost of design modifications (if any).
- Analytical modeling is very useful and popular to support the selection of a dependable architecture for a computer system during the design phase. Also, once the system architecture is selected, detailed modeling provides a powerful tool for the evaluation of the dependability measures (availability, reliability, etc.) of the system under consideration (in development or in operation). In the latter case, modeling needs the support of fault injection and field measurement.
- Field measurements provide helpful support for understanding real phenomena (information on actual error/failure behavior and on possible system bottlenecks) and for quantifying dependability measures. Even if there is no better way to understand the system dependability than by field measurement, analysis of field data can be performed only when the system is already in operation, which could be considered as too late as only little improvements can be performed at this stage in case of identification of any problem. Indeed, feedback from field data from previous systems is very helpful for the development of current products. Measurements provide good support for system reliability improvement.
- Fault injection on a prototype system is usually performed during system validation. It provides valuable information on specific behaviors of the system (or components of the system) in presence of faults. In particular, it allows understanding of the effects of injected faults on the target system and the evaluation of the efficiency of fault-tolerance mechanisms (see e.g., [Arlat et al 1993]. Fault injection is recommended for newly developed systems or for Commercial-Of-The-Shelf (COTS) components for which no (or not enough) dependability information is available from the field.

This paper elaborates on the work presented in [Laprie *et al.* 1998] and [Arlat *et al.* 2001]. Emphasis is put on the first three sets of techniques, namely ordinal evaluation, analytical modeling and field measurements, whose main features are outlined in the next three sections.

1. Ordinal evaluation

Ordinal evaluation methods can be broadly divided into two main categories constructed either from the causes to reach the effects, so as to determine the consequences on the system of component failures or — from the effects to the causes —, to identify causes at the component level of system failures. The two main approaches representative of these two families are respectively: i) Failure Mode, Effects (and Criticality) Analysis — referred to as FME(C)A, and the Fault Tree method, equally known as Cause or Defect Trees.

The exact meaning of "criticality of a failure mode" is context dependent. Indeed, the criticality of a failure mode depends on numerous parameters: severity (or gravity) of the failures considered, probability of the latter's occurrence, possibility of detecting early signs, urgency of the conservatory measures needed, etc. Of these parameters, the first two have become very popular as they allow criticality levels to be defined as couples (e.g., gravity, probability of occurrence). Note as well that the product — or any similar operation — between these two magnitudes to "compute" criticality should not be effected: indeed, gravity is an ordinal magnitude whose values only make sense relative to each other, while a probability is a quantitative magnitude endowed with absolute meaning. System criticality is then defined as the highest criticality level of its failure modes: for each failure mode listed in accordance with severity levels, the maximum permitted probability of occurrence of a failure corresponds, therefore, to a criticality less than or equal to that of the system, equality being usually secured for the most severe failure modes (and even systematically secured, when the most severe failure mode corresponds to the occurrence of catastrophic failures whose consequences are in no way commensurate with less severe failures).

The rest of this section describes the FME(C)A and Fault Tree techniques.

1.1 FME(C)A

FME(C)A [CEI 1985, Leveson 1991] is an inductive approach whose basic principle is to analyze, for each component, the consequences of possible errors so as to identity systematically all the component failure modes as well as the consequences of these failures for the system.

Usually, FMECA relies on the functional structure of the system and makes it possible to highlight design weaknesses "if any" in the system with respect to safety. It can address all design stages, or even an operational system but it is often preferable to apply it as soon as possible during the life cycle to minimize the costs of design modifications. However, its use at an early stage in the development process is rather delicate, as the lack of precise knowledge of the real structure of the system and of the failure modes, etc. becomes a major hurdle.

The general principle used for the application of the method consists in including the following items in a table — after listing the various failure modes based on the functional or structural description of the system—, and for each failure mode of each component:

- its possible causes;
- its effect: it can only act on the behavior of the component itself (local effect), or propagate up to system level (global effect);
- the means of detection that can be employed;
- corrective actions to be implemented, more particularly, when dealing with a catastrophic failure mode;
- criticality of the failure mode: this aspect is not always taken into account yet, being simply referred to as FMEA.

When iterating the application of the method on a subcomponent, the omission of certain failure modes of the component to which it belongs may be revealed. Indeed, the failure modes of the higher-level component appear as combinations of subcomponents' failure modes. In particular, the latter having global effects must necessarily correspond to failure modes of the higher-level component.

The tables summarizing FMECA are useful for the design, as they support certain choices and permit the early detection (and modification) of several drawbacks. They equally provide a valuable medium during system validation because they highlight the salient features to be tested and allow a cause, an elementary failure mode of the component to be associated with an effect when a system failure is reported. Nevertheless, the FMECA-based approach features three main limitations:

- it addresses the failure modes one after the other (to avoid the combinatorial explosion of the analysis), thereby failing to factor in multiple failures (whose importance is no longer in need of a demonstration);
- it supposes that all failure modes of the system components have been determined (which may be problematic in practice);

- it calls for a great deal of information to be handled, particularly in the case of complex systems (which are precisely those for which safety is required); the latter point has become less cumbersome thanks to process computerization.

FMECA applied to the software primarily tries to plan the effects of design faults. A software FMECA established on the basis of the specification and preliminary design may turn out to be very fruitful and at various levels of the life cycle:

- during software development by guiding online error detection or even fault tolerance strategy which has to be selected;
- during software validation, by focusing the tests on certain modules or critical failure modes;
- during software operation and maintenance by facilitating on the one hand, the definition of monitoring procedures and on the other, the understanding of the origin of a failure observed in operation.

Although applications of FMECA to the software are difficult to estimate in practice because only a limited number of papers have been published on this subject, these methods are potentially interesting given their relative cost with respect to software development cost, and the lessons that can be drawn as to the attention which will have to be paid, during validation, to components identified as critical or to the online error detection or even fault tolerance strategy.

1.2 Fault trees

This is a deductive technique that allows combinations of events likely to lead to an undesirable event, such as a catastrophic failure, to be traced. Fault Trees frequently complement the FMECA. Thus, combinations of failures that had been overlooked by FMECA can be considered. Fault Tree may also be easier to run in the early phases of the development process and can be implemented as soon as the requirement analysis has been completed to represent, at a high level of abstraction, the different scenarios of occurrence of an undesirable event likely to affect dependability.

A Fault Tree consists in two successive levels of events connected by gates ("AND", "OR" logic operators). Each event at the output of a given gate is obtained by combining the events located at the input to this gate. The undesirable event analyzed is the tree root. The Fault Tree construction principle relies on breaking down each event encountered, starting from the root event, up to events considered elementary. An event can be regarded as elementary when it is either independent of the others or when its probability of occurrence can be estimated, or simply when one does not intend to, or cannot, break it down further.

The Fault Tree computes the *minimum "cuts*:" for its operation: a cut is a set of events that can lead to the undesirable event at the tree root; a cut is said to be minimal when it contains no other. The study of these minimal cuts permits to highlight the critical events relative to the occurrence of the undesirable event.

With respect to software design fault, Fault Trees have been employed in the first phases of software development [Hourtolle 1987], to detect critical software functions, avoid specification faults and guide the implementation of fault tolerance techniques [Leveson 1991]. Thus, a Fault Tree is constructed for each catastrophic failure. It allows identification of the software critical functions and its critical failure modes (events contained in minimal cuts) for which software fault tolerance techniques have to be implemented.

2. Analytical Modeling

Analytical modeling relies on the description of the system behavior taking into account failure and repair of hardware and software system components and interactions between them. Measures of dependability are assessed by allocating stochastic probabilities or rates to model parameters. Analytical models have long been recognized as a determining factor for rational decision making when considering different possible architectures or maintenance policies during the design of hardware fault-tolerant systems.

Given the ease of modeling they provide, particularly with respect to stochastic dependencies between system components, state-space models constitute the prevailing type of model for evaluating dependability measures. Markov chains are the most commonly used state-space models to model system dependability as they also allow evaluation of various measures related to dependability and performance (i.e., performability measures) based on the same model, when a reward structure is associated to them. The resulting model is referred to as Reward Markov model

To facilitate the generation of large state-space models, high-level specification languages such as Generalized Stochastic Petri Nets (GSPN) and their off-springs are generally used. In particular, the association of a reward structure leads to Generalized Stochastic Reward Petri Net (GSRPN) that can be

automatically converted to Reward Markov models [Trivedi *et al.* 1994]. GSRPNs allow a compact representation of the behavior of systems involving synchronization, concurrency and conflict phenomena. Also, they provide means for structural verification of the model.

Evaluation is achieved in three main closely related phases:

- The choice of the dependability measures to be evaluated.
- The *construction* of one (or several) model(s) describing the behavior of the system.
- The *processing* of the model(s) to evaluate dependability measures.

In the following, the most salient trends related to the choice of dependability measures and to model construction are briefly described as numerous software packages have been devised over the last twenty years to assist model processing (e.g., see [Trivedi *et al.* 1994]). Surveys of the problems related to techniques and tools for dependability and performance evaluation can be found for example in [Reibman & Veeraraghavan 1991] and [Trivedi *et al.* 1994].

2.1 Dependability Measures to be Evaluated

Dependability covers a wide range of measures: reliability, availability, maintainability, safety, etc.. The measures to be evaluated greatly depend on the field of application of the computing system considered (for example: availability for a telecommunication system, reliability for a space probe, safety for the on-board control in a transportation system, etc.). To identify dependability measures, the behavior of a computing system can be schematically depicted by taking into consideration two states of the service delivered: proper and improper. Transitions between these states are governed by the *failure* processes (from a proper service to an improper one) and the *restoration* processes (from an improper service to a proper one).

The main measures are aimed at characterizing the time of proper service delivery. Two main categories of measures are distinguished [Laprie 1995]:

- Measures that characterize the sojourn time in the state where the proper service is being delivered (before reaching the improper service state): these correspond for example to reliability and MTTF that measure the time of proper service delivery prior to a failure.
- Measures that characterize the delivery of proper service with respect to the alternation of proper and improper services: these encompass the various forms used to measure availability (time instant, interval-of-time, or asymptotic).

Most current computing systems feature several performance levels and thus, several modes of services (proper and improper) can be distinguished. According to the viewpoints considered to evaluate dependability, there exist two main (extreme) cases for which the system features:

- several modes of proper service completion and a single mode of improper service;
- a single mode of proper service delivery and several modes of improper service.

A particularly interesting case pertaining to the second category of systems is that of systems exhibiting two modes of improper service following failures with different levels of severity (benign and catastrophic). This allows the measures linked to the evaluation of safety of these systems to be obtained within the very same framework. Thus, safety represents the measurement of time in the safe states (proper service delivery and benign failure) prior to a catastrophic failure. A hybrid measure can be defined that measures the delivery of a proper service relative to the alternation "proper service-improper service" following a benign failure. The advantage of this measure lies in that it allows for the system availability prior to the occurrence of a catastrophic failure to be quantified, and hence, supports the assessment of the usual trade-off between reliability (or availability) and safety [Essamé *et al.* 1997]**.**

2.2. Model Construction

Modeling requires the knowledge of the system architecture (i.e., system composition and interactions between the various components), error detection and fault-tolerance mechanisms (if any) and maintenance policies. At the model level, the associated phenomena are represented by their occurrence rates (failure, repair, error propagation) or conditional probabilities (error detection coverage, recovery coverage, maintenance efficacy).

The main problem posed by the establishment of a Markov chain truly representative of the behavior of a complex system is that of controlling the explosion in number of states. Several techniques have been published to address this problem; they can be grouped into two categories: "largeness avoidance" and "largeness tolerance" techniques [Trivedi *et al.* 1994].

Largeness Avoidance Techniques try to circumvent the generation of very large models. The basic idea is

to construct small sub-models that can be processed in isolation. The results of the sub-models are integrated in a single overall model that is small enough to be processed. From a practical point of view and to the best of our knowledge, most of these techniques are efficient when the sub-models are loosely coupled and become hard to implement when interactions are too complex. Also, largeness avoidance by means of truncation of the least important states (i. e., states with very small probabilities) can be used to complement efficiently largeness tolerance techniques as in [Muppala *et al.* 1992].

The main objective of *Largeness Tolerance Techniques* is to master the complexity of the generation of the global system model through the use of *concise specification methods* and *automated generation* of the model. The specification consists of a set of rules allowing an easy construction of the Markov chain. These rules are based on either a) in-house formalisms or b) well-known formalisms such as GSPNs or their off-springs.

GSPNs and their off-springs appear as a general-purpose approach to the specification and construction of a complex system in a modular way. The basic idea is to generate the model of a modular system by composition of the sub-models of its components; they are referred to as *model composition techniques*. In addition to the GSNP formalism, these techniques make use of composition rules for sub-model interfacing and integration to facilitate model generation, master the complexity and preserve the formalism properties. Several model composition techniques have been published (e.g., see [Meyer & Sanders 1993], [Rojas 1996], [Kanoun & Borrel 1996], [Fota *et al.* 1999b], [Bondavalli *et al.* 1999], [Rabah & Kanoun 1999]) and numerous evaluation tools using GSPNs and their offsprings have been developed.

GSPNs and their off-springs have been used to model real-life systems such as air traffic control systems [Fota *et al.* 1999a, Kanoun *et al.* 1999], space applications [Bondavalli *et al.* 1997] and RAID storage system [Santonja *et al.* 1996].

The evaluation of real-life systems requires the knowledge of numerical values of the model parameters that can be provided either from field data (i.e., failure and repair rates) or from controlled experiments (i.e., coverage factor, various proportions of failure modes). Sensitivity analyses allow identification of the most significant parameters to be estimated from the field or from controlled experiments.

3. Experimental Evaluation

Measuring a real-life system means recording naturally occurring errors and failures in the system while it is running under user workloads. Analysis of such field data can provide valuable information on actual error/failure behavior, quantify dependability measures and identify system bottlenecks. Field measurement involves three main steps: data collection, data validation and data processing.

Data collection consists in the definition of *what* to collect and *how* to collect the data. The kind of data to be collected is directly linked to the kind of behavior to be analyzed and to the quantitative measures to be evaluated to characterize such behavior.

Data validation consists in analyzing the collected data for correctness, consistency, and completeness. This consists in particular in filtering-out invalid data and in coalescing redundant or equivalent data. Usually, the collected data contains a large amount of redundant and irrelevant information, as well as incorrect or incomplete information. Such problems have been observed in several studies, e.g. see [Kaâniche *et al.* 1990, Levendel 1990, Buckley & Siewiorek 1995, Thakur & Iyer 1996]. Thus, preliminary investigation of the data must be performed to classify this information and to facilitate subsequent analyses. Once invalid data is filtered-out and data is coalesced, the basic dependability characteristics of the target system can be identified through data processing.

Data processing consists in performing statistical analyses on the validated data to identify and analyze trends and to evaluate quantitative measures that characterize dependability. Descriptive statistics can be derived from the data to analyze the location of faults, errors and failures among system components, the severity of failures, the time to failure or time to repair distribution, the impact of the workload on the system behavior, the coverage of error detection and recovery mechanisms, etc. Commonly used statistical measures in the analysis include frequency, percentage, probability distribution, and hazard rate function. Basic statistical techniques can be applied to estimate the mean, variance, and confidence intervals of the parameters characterizing these measures (e.g., see [Kendall 1977] for comprehensive statistical methods). More sophisticated analyses can also be performed using trend tests [Kanoun & Laprie 1996] and analytical modeling.

Software faults, and more generally design faults, have become the major dependability bottleneck. This

is confirmed by field data collected on largely deployed systems,, e.g., see [Gray 1990, Moran *et al.* 1990, Cramp *et al.* 1992, Wood 1995]. For example, the analysis of field failures in Tandem computer systems between 1985 and 1990 [Gray 1990] revealed that more than 60% of system failures reported in 1989 were due to software. Accordingly, many experimental studies focused at the analysis of software-related errors. The analysis and modeling of software errors to provide feedback to the development process have been addressed in several papers, e.g., [Musa *et al.* 1987, Lyu 1995, Kanoun *et al.* 1997, Murphy & Levidow 2000]. Several experimental studies have been published to support the analysis of software error characteristics and the modeling of the impact of software failures on dependability, e.g. [Levendel 1990, Kenney & Vouk 1992, Kaâniche *et al.* 1994, Chillarege *et al.* 1995]. The issues addressed in the above mentioned work include: i) categorization of software errors; ii) monitoring of software processes and products through the use of trend tests and statistical quality control, and iii) evaluation of quantitative measures characterizing the software failure intensity and time to failure using reliability growth models.

Analyses of the software behavior, based on the observed failure data, allow identification of possible weakness. Once identified, these weaknesses can be adduced. Reliability analyses help the manager to anticipate the software behavior. Using results drawn to past experience, (s)he can plan more efficiently the development process. In this context, a program of software reliability improvement is an element of a more general software process maturity effort.

In addition to specific experiences, several papers and books have already been published advocating and defining methods for improving software process based, among other things, on data collection (see e.g., [Musa 1998; Kanoun 2001]).

One of the most common objections to software reliability programs is their cost. The relationship between the level of dependability required and the associated cost is a complicated one as it includes such factors as the supplier's rework cost, the maintenance cost and the failure consequences to the user. However, past experience has shown the cost of fixing a fault uncovered during operation to be at least one order of magnitude higher than the cost of the same fault detected during development. When the costs to the user and the negative impacts to the producer reputation are included the effect is magnified.

The benefits from a reliability improvement program sometimes cannot be perceived in a single product lifecycle. In that case, the cost of a reliability program has to be regarded as an investment for subsequent systems rather than as an overhead for a single system. Usually the gains are substantial — even if they are not always immediately felt. It is worth noting that all the companies that have followed a well-defined program for improving software process and quality agree on the fact that the benefits are worthwhile. However, it is very difficult to partition the gains according to the methods used (e.g., the relative impact of fault prevention and fault removal techniques is very difficult to be assessed).

By way of example, the results obtained through the quality program started at AT&T's International DEFINITY PBX [Donnelly *et al.* 1992], based, among other things, on software reliability evaluation show reduction factors of:

- 10, in customer-reported problems.
- 10, in maintenance cost.
- 2, in the test interval.
- 3, in new product introduction interval.

It can be argued that the reported examples concern large and well-established companies working most of the time on large software projects. This is true. However, more recently published experiences show that following well-organized measurement programs is also worthwhile for small organizations (see e. g., [Kautz 1999] or [Grable *et al.* 1999]). Nevertheless, the data collection program should be adjusted to small companies to suit the company products, goals and means.

4. Conclusion

This paper presented the state-of the-art in dependability evaluation based on ordinal analyses, analytical modeling and field measurements. We have focused on salient trends for each technique considered alone. Indeed, system dependability is usually evaluated based on the combined use of all the presented methods that are advantageously complemented by fault injection techniques.

References

[Arlat et al. 1993] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, “Fault Injection and Dependability Evaluation of Fault-Tolerant Systems”, *IEEE Trans. on Computers,* 42 (8), pp.913-23, 1993.

[Arlat et al. 2001] J. Arlat, K. Kanoun, H. Madeira, J. V. Busquets, T. Jarboui, A. Johansson and R. Lindström, "State of the Art", Available at http://www.laas.fr/dbench/delivrables.html, DBench Project IST 2000-25425 Deliverable, N°CF1, September 2001 (Also LAAS Report 01-605).

[Bondavalli et al. 1997] A. Bondavalli, I. Mura and M. Nelli, "Analytical Modeling and Evaluation of Phased Mission Systems in space Applications", 2nd IEEE High Assurance System Engineering Workshop (HASE), (Bethesda, MD, USA), pp.85-91, 1997.

[Bondavalli et al. 1999] A. Bondavalli, I. Mura and K. S. Trivedi, "Dependability Modeling and Sensitivity Analysis of Scheduled Maintenance Systems", in 3rd European Dependable Computing Conference (EDCC-3), (Prague, Czech Republic), pp.7-23, Springer, 1999.

[Buckley & Siewiorek 1995] M. F. Buckley and D. P. Siewiorek, "VAX/VMS Event Monitoring and Analysis", in Proc. 25th Int'l Symposium on Fault-Tolerant Computing (FTCS-25), (Pasadena, CA, USA), pp.414-23, 1995.

[CEI 1985] *Techniques d'analyse de la fiabilité des systèmes - Procédure d'Analyse des Modes de Défaillance et de leurs Effets (AMDE)*, Commission Électrotechnique Internationale (CEI), Rapport de Normalisation N°812, 1985. *In French*.

[Chillarege et al. 1995] R. Chillarege, S. Biyani and J. Rosenthal, "Measurement of Failure Rate in Widely Distributed Software", in 25th IEEE International Symposium On Fault Tolerant Computing (FTCS-25), (Pasadena, CA, USA), pp.424-33, 1995.

[Cramp et al. 1992] R. Cramp, M. A. Vouk and W. Jones, "An Operational Availability of a Large Software-Based Telecommunications System", in 3rd Int. Symp. on Software Reliability Engineering, (RTP, NC, USA), pp.358-66, 1992.

[Donnelly *et al.* 1992] M. M. Donnelly, J. D. Musa, W. W. Everett and G. Wilson, *Best Current Practice: Software Reliability Engineering*, AT&T Bell Laboratories Software Quality and Productivity Cabinet, N°45370B-930326-01TM, October 1992.

[Essamé et al. 1997] D. Essamé, J. Arlat and D. Powell, "Available Fail-Safe Systems", in 7th Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), (Tunis, Tunisia), pp.176-82, 1997.

[Fota et al. 1999a] N. Fota, M. Kâaniche and K. Kanoun, "Dependability Evaluation of an Air Traffic Control Computing System", Performance Evaluation, 35 (3-4), pp.553-73, 1999.

[Fota et al. 1999b] N. Fota, M. Kâaniche and K. Kanoun, "Incremental Approach for Building Stochastic Petri Nets for Dependability Modeling", in Statistical and Probabilistic Models in Reliability pp.321-35, Birkhäuser, 1999.

[Grable *et al.* 1999] R. Grable, J. Jernigan, C. Pogue and D. Divis, "Metrics for Small Projects: Experiences at the SED", *IEEE Software*, March/April, pp.21-9, 1999.

[Gray 1990] J. Gray, "A Census of Tandem System Availability Between 1985 and 1990", IEEE Trans. on Reliability, R-39 (4), pp.409-18, 1990.

[Hourtolle 1987] C. Hourtolle, *Conception de logiciels surs de fonctionnement : analyse de la sureté des logiciels - mécanismes de décision pour la programmation en N-versions,* Thèse de Docteur-Ingénieur, INP, Toulouse, October 1987.

[Kaâniche et al. 1990] M. Kaâniche, K. Kanoun and S. Metge, "Failure Analysis and Validation of a Telecommunication Equipment Software System", Annales des Telecommunications, 45 (11-12), pp.657-70, 1990.

[Kaâniche et al. 1994] M. Kaâniche, K. Kanoun, M. Cukier and M. Bastos Martini, "Software Reliability Analysis of Three Successive Generations of a Switching System", in First European Conference on Dependable Computing (EDCC-1), (Berlin, Germany), LNCS, 852, pp.473-90, Springer, 1994.

[Kanoun & Borrel 1996] K. Kanoun and M. Borrel, "Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions between Hardware and Software Components", in IEEE Inter. Computer Performance & Dependability Symposium (IPDS'96), (Urbana-Champaign, IL, USA), pp.252-61, 1996.

[Kanoun & Laprie 1996] K. Kanoun and J.-C. Laprie, "Trend Analysis", in Handbook of Software Reliability Engineering (M. Lyu, Ed.), pp. 401-37 (Chapter 10), McGraw Hill, 1996.

[Kanoun 2001] K. Kanoun, "A Measurement-Based Framework for Software Reliability Improvement", *Annals of Software Engineering*, 11 (1), pp.89-106, November 2001.

[Kanoun et al. 1997] K. Kanoun, M. Kaâniche and J.-C. Laprie, "Qualitative and Quantitative Reliability Assessment", IEEE Software, 14 (2), pp.77-86, 1997.

[Kanoun et al. 1999] K. Kanoun, M. Borrel, T. Moreteveille and A. Peytavin, "Modeling the Dependability of CAUTRA, a Subset of the French Air Traffic Control System", IEEE Transactions on Computers, 48 (5), pp.528-35, 1999.

[Kautz 1999] K. Kautz, "Making Sense of Measurement for Small organizations", *IEEE Software*, March/April, pp.14-20, 1999.

[Kendall 1977] M. G. Kendall, The Advanced Theory of Statistics, Oxford University Press, 1977.

[Kenney & Vouk 1992] G. Q. Kenney and M. A. Vouk, “Measuring the Field Quality of Wide-Distribution Commercial Software”, in 3rd IEEE Int. Symposium on Software Reliability Engineering (ISSRE'92), (Raleigh, NC, USA), pp.351-7, 1992.

[Laprie 1995] J.-C. Laprie, “Dependable Computing: Concepts, Limits, Chalenges”, in 25th International Symposium on Fault-Tolerant Computing, (Pasadena, CA, USA), pp.42-53, 1995.

[Laprie et al. 1998] J.-C. Laprie, J. Arlat, J.-P Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.-C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, J.-C. Laprie, C. Mazet, D. Powell, C. Rabejac and P. Thévenot, “Dependability Handbook”, LAAS report No 98346, August 1998. English version of “Guide de la sûreté de fonctionnement”, Cépaduès Editions, Second edition, 1996.

[Levendel 1990] Y. Levendel, “Reliability Analysis of Large Software Systems: Defects Data Modeling”, IEEE Transactions on Software Engineering, SE-16 (2), pp.141-52, 1990.

[Leveson *et al.* 1991] N. G. Leveson, S. S. Cha and T. J. Shimeall, “Safety Verification of ADA Programs Using Software Fault Trees”, *IEEE Software* (7), pp.48-59, 1991.

[Lyu 1995] M. R. Lyu (Ed.), Handbook of Software Reliability Engineering, McGraw-Hill, 1995.

[Meyer & Sanders 1993] J. F. Meyer and W. H. Sanders, “Specification and Construction of Performability Models”, in Int. Workshop on Performability Modeling of Computer and Communication Systems, (Mont Saint Michel, France), pp.1-32, 1993.

[Moran et al. 1990] P. Moran, P. Gaffney, J. Melody, M. Condon and M. Hayden, “System Availability Monitoring”, IEEE Trans. on Reliability, R-39 (4), pp.480-5, 1990.

[Muppala et al. 1992] J. K. Muppala, A. Sathaye, R. Howe, C and K. S. Trivedi, “Dependability Modeling of a Heterogeneous VAX-cluster System Using Stochastic Reward Nets”, in Hardware and Software Fault Tolerance in Parallel Computing Systems (D. R. Avresky, Ed.), pp.33-59, 1992.

[Murphy & Levidow 2000] B. Murphy and B. Levidow, “Windows 2000 Dependability”, in Workshop on Dependable Networks and Operating Systems, at the Int’l Conference on Dependable Systems and Networks (DSN-2000), (New York, NY, USA), pp.D20-D8, 2000.

[Musa 1998] J. Musa, *Software Reliability Engineering,* 391p., Computing McGraw-Hill, 1998.

[Musa et al. 1987] J. Musa, A. Iannino and K. Okumoto, Software Reliability: Measurement, Prediction, Application, Computer Science Series, 621p., McGraw-Hill, New-York, 1987.

[Rabah & Kanoun 1999] M. Rabah and K. Kanoun, “Dependability Evaluation of a Distributed Shared Memory Multiprocessor System”, in 3rd European Dependable Computing Conference (EDCC-3), (Prague, Czech Republic), pp.42-59, Springer, 1999.

[Reibman & Veeraraghavan 1991] A. Reibman and M. Veeraraghavan, “Reliability Modeling: An Overview for System Designers”, IEEE Computer, April, pp.49-57, 1991.

[Rojas 1996] I. Rojas, “Compositional Construction of SWN Models”, The Computer Journal, 38 (7), pp.612-21, 1996.

[Santonja et al. 1996] V. Santonja, M. Alonso, J. Molero, J. J. Serrano, P. Gil and R. Ors, “Dependability Models of RAID Using Stochastic Activity Networks”, in 2nd European Dependable Computing Conference (EDCC-2), (Taormina, Italy), pp.141-58, Springer, 1996.

[Thakur & Iyer 1996] A. Thakur and R. K. Iyer, “Analyze-NOW — An Environment for Collection & Analysis of Failures in a Network of Workstations”, IEEE Transactions on Reliability, 45 (4), pp.561-70, 1996.

[Trivedi et al. 1994] K. S. Trivedi, B. R. Haverkort, A. Rindos and V. Mainkar, “Methods and Tools for Reliability and Performability: Problems and Perspectives”, in Proc. 7th Int'l Conf. on Techniques and Tools for Computer Performance Evaluation (G. Kotsis, Ed.), LNCS, 794, pp.1-24, Springer, 1994.

[Wood 1995] A. Wood, “Predicting Client/Server Availability”, Computer (April), pp.41-8, 1995.