DATA COLLECTION FOR SOFTWARE RELIABILITY DATABASES

Karama Kanoun

2nd International Coference on Reliability, maintainability and Safety (ICRMS), Pékin (Chine), 7-10 juin 1994, pp. 3-10.
Invited talk

DATA COLLECTION FOR SOFTWARE RELIABILITY DATABASES

Karama KANOUN
LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse - FRANCE
e-mail: kanoun@laas.fr

INTRODUCTION

The need to meet competing constraints while reducing the system life-cycle cost has made people realize a) that dependability is vital for the whole life-cycle of a product and b) that it is fundamental to include qualitative and quantitative dependability aspects from the early phases of the life-cycle. Achieving dependability requires painstaking efforts and a high level of commitment of all those involved in the design and production, from specifications to operation and maintenance. A fully integrated and systematic approach is needed to ensure that dependability is taken into account as early as the specification phase, and correctly handled during development and that the goals are reached for the final product.

One of the major objections to dependability is its cost which usually increases with the level required. Indeed the relationship between the level of dependability required (or achieved) and the associated cost is further complicated when taking into account other factors such as the supplier's rework cost or the maintenance cost and that of the consequences of failures, for the customer. Figs. 1 and 2 (which are extracted from [4]) respectively give examples of the relationship between the required level of reliability and a) the production cost as felt by the supplier, b) the life-cycle cost as felt by the customer. It is worth noting that from both points of view a minimum cost does exist at a particular level of dependability. Generally these two minimum costs do not correspond to the same level and the suppliers should therefore aim to reduce the customer's life-cycle cost rather than their own production costs. Thus a tradeoff has to be reached.

O'Conner [26] goes even further by stating that the curves of Figs. 1 and 2 correspond to a "traditional view" and that, from a "modern view", the total costs continue to reduce indefinitely as reliability is improved. This viewpoint is supported by the fact that the cost of reliability has to be regarded as an investment for subsequent systems rather than as an overhead for the considered system. Usually the gains are substantial — even if they are not always achieved immediately. For example, the results presented in [7] show that the rework costs have shrunk to a quarter of their original value after completion of a five year program aimed at process improvement within the Raytheon Equipment Division (this study was carried out over 15 projects).

Learning from past experience is one of the fundamentals for dependability (and product) improvement: feedback from field experience is of prime importance. This is made possible by **data collection** on several products in development or in the field (when possible) aimed at **database** creation. For hardware, such databases have been built up over several decades allowing establishment of **standards** relative to dependability. For software, such standards are not available for the moment despite several attempts made by national or international standardization groups. This presentation deals with data collection and database construction for software dependability.

ABSTRACT

This paper addresses the problem of data collection and database construction for software dependability. It is composed of four sections. The first section investigates the lack of software dependability standards. The second section defines the kind of data to be collected and the associated data collection organization enabling software dependability databases to be created. The last two sections deal with the types of analysis and processing to be carried out on these data, and examples of results that can be obtained.

INTRODUCTION

The need to meet competing constraints while reducing the system life-cycle cost has made people realize a) that dependability is vital for the whole life-cycle of a product and b) that it is fundamental to include qualitative and quantitative dependability aspects from the early phases of the life-cycle. Achieving dependability requires painstaking efforts and a high level of commitment of all those involved in the design and production, from specifications to operation and maintenance. A fully integrated and systematic approach is needed to ensure that dependability is taken into account as early as the specification phase, and correctly handled during development and that the goals are reached for the final product.

One of the major objections to dependability is its cost which usually increases with the level required. Indeed the relationship between the level of dependability required (or achieved) and the associated cost is further complicated when taking into account other factors such as the supplier's rework cost or the maintenance cost and that of the consequences of failures, for the customer. Figs. 1 and 2 (which are extracted from [4]) respectively give examples of the relationship between the required level of reliability and a) the production cost as felt by the supplier, b) the life-cycle cost as felt by the customer. It is worth noting that from both points of view a minimum cost does exist at a particular level of dependability. Generally these two minimum costs do not correspond to the same level and the suppliers should therefore aim to reduce the customer's life-cycle cost rather than their own production costs. Thus a tradeoff has to be reached.

O'Conner [26] goes even further by stating that the curves of Figs. 1 and 2 correspond to a "traditional view" and that, from a "modern view", the total costs continue to reduce indefinitely as reliability is improved. This viewpoint is supported by the fact that the cost of reliability has to be regarded as an investment for subsequent systems rather than as an overhead for the considered system. Usually the gains are substantial — even if they are not always achieved immediately. For example, the results presented in [7] show that the rework costs have shrunk to a quarter of their original value after completion of a five year program aimed at process improvement within the Raytheon Equipment Division (this study was carried out over 15 projects).

Learning from past experience is one of the fundamentals for dependability (and product) improvement: feedback from field experience is of prime importance. This is made possible by **data collection** on several products in development or in the field (when possible) aimed at **database** creation. For hardware, such databases have been built up over several decades allowing establishment of **standards** relative to dependability. For software, such standards are not available for the moment despite several attempts made by national or international standardization groups. This presentation deals with data collection and database construction for software dependability.

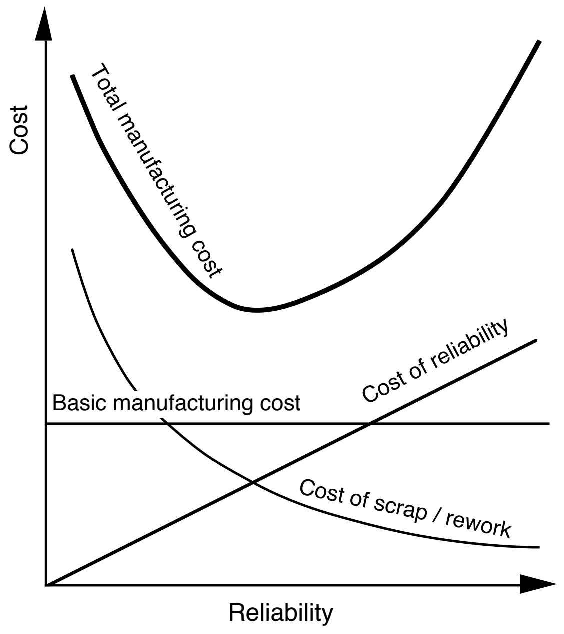


Fig. 1 Cost and reliability for the supplier

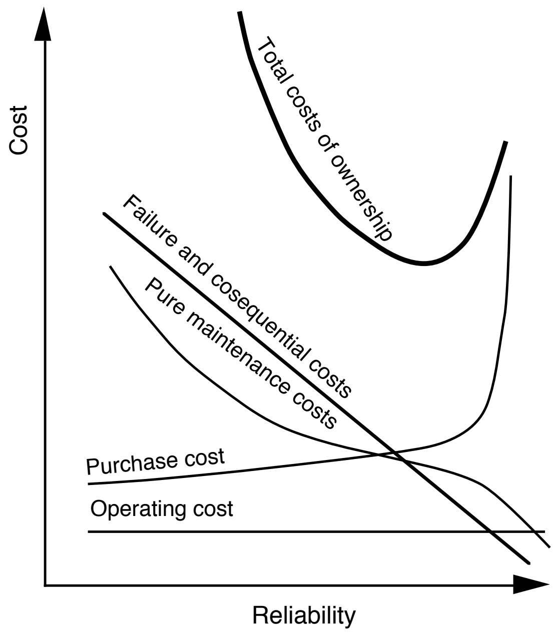


Fig. 2 Cost and reliability for the customer

The paper is composed of four sections. The first section investigates the lack of software dependability standards. The second section defines the kind of data to be collected and the associated data collection organization enabling software dependability databases to be created. The last two sections deal with the types of analysis and processing to be carried out on these data, and examples of results that can be obtained.

SOFTWARE DEPENDABILITY DATABASES: PROBLEMS AND OBSTACLES

Building up a database is a long-term process achieved in several steps: definition of the goals and the data to be collected, data collection, data analysis and processing. Analysis may have either near-term or long-term objectives. Near-term results provide immediate and efficient feedback to the considered system: they help improve the quality of the product and its development process. It may be some time however before the benefits begin to be felt (i.e., long-term results); in this case feedback may be only beneficial to the subsequent generations of the product or to broadly similar products. It allows thus accumulation of thorough knowledge of the developed systems in order to a) improve their development (i.e., process maturity) and b) better predict their behavior.

In the early phases of a collection it is important to clearly state the main objectives: a collection program which is ideal for certain objectives may turn out to be ill-suited for others. These objectives depend themselves on the point of view adopted (the supplier or the customer) as well as on the life-cycle phase considered (development, operation, maintenance).

- During development, the supplier is interested in:
  - estimating of the number of faults[1] in the software so as to plan the test effort needed to remove them,
  - managing the development so as to plan exit from one phase to the following one,
  - managing the software configurations,
  - producing easily maintainable software through the use of complexity measures, etc.
- When the software is in operation:

[1] The terminology we shall be using is that defined in [19].

- the supplier is interested in the estimation of the expected number of failures among all installations (or the number of corrections to be performed on the software) in order to estimate the maintenance effort still needed,
- the customer is concerned with the mean time to failure or failure rate (either the instantaneous failure rate or the expected residual failure rate in operational life) so as to evaluate the reliability of the whole system (hardware and software).

Each objective may be composed of several sub-objectives; for example, considering the objective "reliability evaluation, one may be interested in a) the failure rate of the whole software, b) the failure rates of all or some of the components and or c) according to some failure modes.

Among the main problems and difficulties which are inherent to the software and its validation methods are:

- the fact that a piece of software is regarded as an intellectual product and is, as thus, related to the coding, verification and validation staff,
- the piece of software is the result of the combined use of coding and verification and validation methods and tools; in other words, it is related to the production process which, in turn, depends on the nature of the application (e.g., critical or non-critical),
- production processes may vary according to the companies concerned; thus the results achieved in one company may not necessarily be valid for the others.

The nature and type of data to be collected are strongly related to the objectives considered and the kind of data processing that can be carried out. Actually, data collection assumes the existence of methods and models allowing the collected data to be processed. However, no method or model has been shown to be effective for all the objectives cited above. Moreover, due to the corrections performed on the software, the failure process is not repetitive leading to the problem of validation of the statistical processing results. In addition, by its nature, data collection itself implicitly makes assumptions about the nature of the failures or other processes.

The above considerations support the view that it is very difficult to formulate general standards for software dependability similar to existing hardware standards (see e.g., MIL-HDBK 217). However, establishing databases for a given company, developing software for similar applications and using similar development methods is not an insurmountable task. The use of this type of database is common practice for software suppliers like AT&T Bell Laboratories [25, 11, 21], Hewlett Packard [9], IBM [6, 8], Jet Propulsion Laboratory [5], Raytheon [7][2], etc. These databases help the companies to improve the maturity of their software processes [27], which in turn help them improve the quality / reliability of the developed software products at no extra cost.

Another problem when trying to establish general standards is data confidentiality: many companies have their own database related to development or / and operation but are not prepared to disclose them as they may be used by competing companies to gain a competitive edge.

In face of such objectives and difficulties, there are only a few methods and models characterizing the software and its behavior and allowing database construction. Among these are the following:

- cost models as those derived in [3],
- complexity measures, such as McCabe's cyclomatic number [23], statement count, or data flow complexity; usually interesting correlations are derived; however no formal link between these measures and reliability on one hand and between these measures and maintainability on the other hand has been found,

[2] Note that the list of references given above is not exhaustive at all, they are cited to give an idea about the work carried out in this field.

- empirical studies trying to relate software fault density to size (this is very helpful input during development) [1], or to link the software failure rate to some other parameters (related to software or environment characteristics) [24],
- trend tests for reliability growth or decrease detection (based on the statistical processing of failure data), allowing development management through follow up of reliability evolution,
- reliability evaluation using reliability growth models.

The latter two methods (trend analysis and reliability evaluation) form the core of the approaches to software dependability and involve the same type of data to be collected. Actually, reliability growth model application may and should be based on trend analysis results so as to improve their outputs [17]. Data collection with the view of trend analysis and reliability evaluation is addressed in the next section.

DATA COLLECTION, DEFINITION AND ORGANIZATION

This section is devoted to a) the definition of data to be collected and b) the practical organization of data collection and some recommendations relative to the setting up of a data collection program.

1. Data To Be Collected

A tradeoff between the amount of data to be collected and the subsequent effort has to be found: a complete and detailed data collection program may help better understand the phenomena involved but the magnitude of effort required may be discouraging for the staff concerned. Data to be collected is of two types:

- data characterizing the product, the production process, and the usage environment: software size, language, functions, current version, verification and validation methods and tools used, load, etc., this being referred to as background information in [9],
- data relative to failures and corrections: date of occurrence, nature, consequence, fault location, etc.

Usually data are collected through use of failure and correction forms featuring well-defined headings that have to be filled in either by the staff member who observed the failure or by the person in charge of handling modifications. The forms must be constructed in such a way that the information given is well structured and easy to enter, a large part of it being composed of short tick-off questions. There forms may be filled in manually or automatically. Bases for good data collection procedures can be found in [2].

By way of example, the type of information to be entered in these forms depending on data collection objectives is as follows:

- the consequences of failures, in order to follow up software reliability with respect to certain failure modes or some critical tasks,
- fault location on the correction report, in order to study the reliability of the components or of some of them; this is very useful for component re-use which is one of the main concerns of the industry now,
- the environmental conditions under which failures occur, in order to investigate the influence of the load on software behavior, etc.

Also note that, during the earlier validation phases, it may be impractical to fill in a failure report and a correction report for each failure and correction, due to the large number of failures that may occur. Thus, data may first be collected in the form of "number of failures" per unit of time. The unit of time may initially be the day and later be increased to the week or even a longer period of time, according to the number of failures observed. The information derived from such data collection and analysis is limited in scope but may be useful for these early phases.

2. Data Collection Organization

Data to be collected must be defined according to the goals that were assigned to the study. Also, a prerequisite for a successful collection of data is the clear definition of the role assigned to each staff member involved in data collection from the project manager to the validation team and users, etc. In addition, motivation and training are key factors, people should be briefed about the objectives of data collection and a quick feedback should be aimed at (even if incomplete initially for lack of information), in order to motivate people. Likewise, it should be clear to everybody that data will not be used to assess the performance of the programmer or the validation team but for product or/and the production process management and reliability follow up. The staff will be all the more ready to accept the time overhead incurred during data recording as they are convinced of its necessity and potential benefit, either immediately or later.

Data collection must also to be included in the development process and must not be considered on its own. For example, when other data collection procedures are already in place for other purposes (such as configuration management), one has to try to merge them by asking for more information in the same collection form for example. Again a clear re-definition of the goals and report forms is needed.

Setting up a dependability data collection — a priori — entails cost overheads, mainly during development which may be an obstacle. However as we stated in the introduction, the gain brought about by the analysis of data by far compensates this overhead. For example, past experience has shown the cost of a fault detected during operation to be 100 times higher to the cost of the same fault detected during development [3].

DATA VALIDATION, ANALYSIS AND PROCESSING

Usually the collected data include reports corresponding to actual software failures as well as extraneous data such as false reports. Filtering the raw data is therefore needed in order to keep only those items corresponding to genuine software faults. Based on our experience [12] as well those reported in [2] and [21], the ratio between validated data and raw data approximates 50%. This step is essential and has to be carefully carried out since subsequent processing (which consists of: a) trend analysis to follow up reliability evolution and b) reliability evaluation, if needed) is carried out on the validated data.

1. Trend Analysis

Reliability growth can be analyzed through use of trend tests which give a better insight into the evolution of reliability. There are a number of trend tests which can be utilized to help determine whether the system undergoes reliability growth or decrease. These tests can be grouped into graphical and analytical tests. Graphical tests consist of plotting some observed failure data such as the inter-failure times or the number of failures per unit of time versus time in order to visually derive the trend displayed by the data: as such they are informal. Note that graphical tests are of common practice in the industry [28]. On the other hand, analytical tests correspond to more elaborate procedures as they are of statistical nature: the raw data are processed in order to derive trend factors. We recommend the Laplace test whose interpretation has been modified to detect reliability growth (respectively decrease) on average and local trend change [15]. It consists of evaluating a statistical indicator called Laplace factor, $u(k)$, where k is the considered unit of time. Depending on the value and on the evolution of this factor, reliability growth or decrease is derived as shown in Fig. 3.

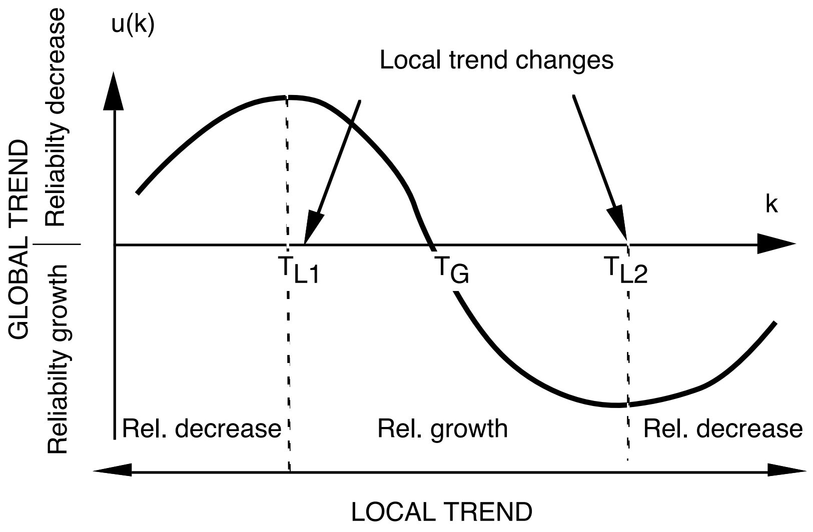


Fig. 3 Laplace factor and reliability evolution

Some typical results that can be drawn from trend analysis are now given.

Reliability decrease at the beginning of a new activity is generally expected and considered as normal. Also, reliability decrease may result from regression faults. Trend analysis allows this kind of behavior to be noticed. If the length of the decrease period seems long[3], it will have to be carefully monitored and, in some situations, if it keeps decreasing it may be indicative of problems: analyzing the reasons of this decrease as well as the nature of the activated faults is then of prime importance in that kind of situations. Such analysis may support the decision to re-examine the corresponding part of the software.

Reliability growth occurring after reliability decrease is usually welcomed since it shows that, following the removal of faults, the corresponding activity reveals less and less faults.

Stable reliability indicates that a) either the software does not undergo corrective maintenance, or b) the corrective actions performed are of no perceptible effect on reliability. When the software is under validation, stable reliability with almost no failures means that the corresponding activity has reached "saturation": the application of the corresponding test sets does not reveal new faults. One can either stop testing or introduce new test sets or proceed to the next phase. More generally, it is recommended to keep applying a test set as long as it exhibits reliability growth and to stop its application when stable reliability with almost no failures is reached. As a consequence, in real situations, the fact that stable reliability is not reached may lead the validation team (and the manager as well) to make the decision to continue testing before software delivery since it will be more efficient to continue testing and removing faults in the validation phase rather than in operation.

Finally, trend analyses may be of great help for reliability growth models as they provide more accurate estimations when they are applied to data displaying trend in accordance with their assumptions rather than blindly.

3. Reliability Growth Models

On the last two decades several reliability growth models have been developed (see e.g., [29] for a survey). Depending on the model, the mean time to the next failure, the failure intensity, the cumulative number of failures or/and the residual failure rate can be evaluated. Model execution is in two steps: parameter estimation based on observed failure data and reliability evaluation (replacing the parameters by their estimated values in the dependability measure expressions).

[3] Usually the length of the reliability decrease period is determined by past experience relative to previous products.

Statistical criteria allow a) appreciation of how well the results fit the data and b) comparison of estimations derived from several models. To serve our purpose we have selected four models allowing different kinds of behavior to be modeled: the hyperexponential model (Kanoun-Laprie) [18], the exponential model (Goel-Okumoto) [10], the S-Shaped model (Yamada et al.) [30] and the doubly stochastic model (Littlewood-Verrall) [22]. For each model Fig. 4 gives the expression and evolution of the failure intensity, h(t), or the failure rate, λ(t).

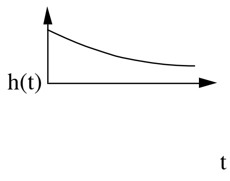
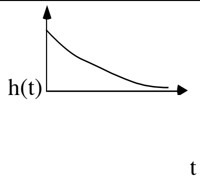
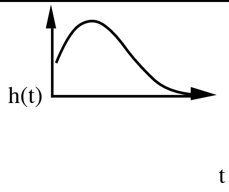
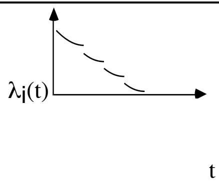
Model	**h(t) or λ(t) shape**
Hyperexponential $h(t) = \frac{\omega\zeta_{sup}e^{-\zeta_{sup}t}+\varpi\zeta_{inf}e^{-\zeta_{inf}t}}{\omega e^{-\zeta_{sup}t}\ \varpi e^{-\zeta_{inf}t}}$	
Exponential $h(t) = N\ \phi\ \exp^{-\phi\ t}$	
S-Shaped $h(t) = N\ \phi^2\ t\ \exp^{-\phi\ t}$	
Doubly Stochastic $\lambda_i(t) = \frac{\alpha}{t+\psi(i)}$ $\psi(i) = \beta_1+\beta_2 i$	

Fig. 4 Some reliability growth models

EXAMPLES OF RESULTS OBTAINED FROM DATA ANALYSIS

Based on the above trend tests and reliability growth models, we have developed a) a method allowing reliability analysis and evaluation [17] and b) a tool for implementing trend tests and reliability growth models: SoRel [16]. SoRel has been used to follow up and evaluate the reliability of several real-life systems. The characteristics of some of these systems are summarized in Table 1. For the validation phase, the main results concern the evolution of reliability in response to debugging activities and the prediction of the number of faults that will be activated over the next periods of time [12]. During operation, the objectives of reliability analysis are more various. In the following are some examples illustrated through the results obtained for the first three systems (which are electronic switching systems, ESS). For the
E10-B, the evaluation of the residual failure rate in operation derived from application of reliability growth models to failure data collected on the software [13] allowed the dependability of the whole ESS to be evaluated (accounting for hardware and software). For the TROPICO-R ESSs we have followed two complementary approaches [14, 17]:

System	Languages	Volume	Observation	Phases	# Systems	# FR and/or CR
E10-B	Assembler	100 k-bytes	3 years	Val. / Op.	1400	58 FR / 136 CR
TROPICO-R 1500	Assembler	300 k-bytes	27 months	Val. / Op.	15	461 CR
TROPICO-R 4096	Assembler	350 k-bytes	32 months	Val. / Op.	42	227 CR
Telecommunication Equipment	PLM-86	$5\ 10^5$ inst.	16 months	Val.	4	2150 FR
Work station	various	--	4 years	Op.	1	414 FR

Val. : validation Op. : operation FR: failure report CR: correction report

Table 1 Some real-life software systems for which reliability has been analyzed using SoRel

- from the supplier's point of view, estimation of the maintenance effort to be made in operation in order to meet the correction reports issued by the various customers,
- from the customer's view point, estimation of the residual failure rate in operation in order to evaluate the impact of software reliability on the ESS reliability.

CONCLUSIONS

Reliability data collection and analysis is only one step in the establishment of a software dependability database within a company. A lot of theoretical and practical work has still to be done before reaching a level where the reliability of a software program can be predicted before its realization using data relative to production process and some assumed characteristics of this software.

The notion of software component re-use makes it urgent to build up internal databases. As previously stated the first step is to collect and analyze reliability data as soon as possible and for as many product components as possible. This process make it possible to evaluate reliability of the various components. Reliability of the whole software may be evaluated through the knowledge of the reliability of each component using a multi-component approach [18, 20].

REFERENCES

[1] V.R.Basili, B.T.Perricone, “Software errors and complexity: an empirical investigation”, *Communications of the ACM*, 27 (1), January 1984, pp. 42-52.

[2] V.R.Basili, D.M.Weiss, "A methodology for collecting valid software engineering data", *IEEE Trans. on Software Engineering,* vol. 10, no 6, Nov. 1984, pp. 728-738.

[3] B.Boehm, *Software engineering economics*, Prentice Hall, Englewood Cliffs, New York, 1981.

[4] British Standard, Reliability of constructed or manufactured products, systems, equipments and components, Part 0: *Introductory guide to reliability*, 1986.

[5] M.Bush, "Improving software quality: the use of formal inspections at the Jet Propulsion Laboratory", Proc. 12th *International Conference on Software Engineering (ICSE12)* , March 26-28, 1990, Nice, France, pp. 196-199.

[6] 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.

[7] R.Dion, "Process improvement and the corporate balance sheet", *IEEE Software*, July 1993, pp. 28-35.

[8] M.Dyer, *The cleanroom approach to quality software development,* John Wiley & Son, Inc., 1992.

[9] R.B.Grady, *Practical software metrics for project management and process improvement*, Hewlett Packard, PTR Prentice Hall, Inc., 1992.

[10] A.L.Goel, K.Okumoto, "Time dependent error-detection rate model for software and other performance measures", *IEEE Trans. on Reliability*, vol. R-28, no 3, 1979, pp. 206-211.

[11] A.Iannino, J.D.Musa, "Software reliability engineering at AT&T", Proc. *PSAM Conference (Probabilistic Safety Assessment and Management)*, Beverly Hills, California, USA, Feb. 1991, pp. 485-491.

[12] M.Kaâniche, K.Kanoun, S.Metge, "Failure analysis and validation monitoring of a telecommunication equipment software system", *Annales des Télécommunications*, vol. 45, no. 11-12, 1990, pp. 657-670, *in French*.

[13] K.Kanoun, T.Sabourin, "Software dependability of a telephone switching system", Proc. 17th *IEEE Int. Symp. on Fault-Tolerant Comp. (FTCS-17),* Pittsburgh, Pennsylvania, July, 1987, pp. 236-241.

[14] 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.

[15] K.Kanoun, J.C.Laprie, "Software reliability trend analyses: from theoretical to practical considerations", LAAS research report, no 93.001, Jan. 1993.

[16] K.Kanoun, M.Kaâniche, J.C.Laprie, S.Metge, "SoRel: a tool for software reliability analysis and evaluation from statistical failure data", Proc. 23th *IEEE Int. Symp. on Fault-Tolerant Comp. (FTCS-23),* Toulouse, France, June, 1993, pp. 654-659.

[17] K.Kanoun, M.Kaâniche, J.C.Laprie, "Experience in software reliability: from data collection to quantitative evaluation", Proc. 4th *International Symposium on Software Reliability Engineering, (ISSRE'93),* Denver, Colorado, Nov. 1993, pp. 234-245.

[18] J.C.Laprie, K.Kanoun, C.Béounes, M.Kaâniche, "The KAT — Knowledge-Action-Transformation — approach to the modeling and evaluation of reliability and availability growth", *IEEE Trans. on Software Engineering*, vol. 17, no 4, April 1991, pp. 370-382.

[19] J.C.Laprie, *Dependability: basic concepts and terminology,* dependable computing and fault-tolerant systems, vol. 5, J.-C. Laprie Editor, Springer Verlag, Wien, New York, 1992.

[20] J.C.Laprie, K.Kanoun, "X-Ware reliability and availability modeling", *IEEE Trans. on Software Engineering*, vol. 2, Feb. 1992, pp. 130-147.

[21] Y.Levendel, "Reliability analysis of large software systems: defect data modeling", *IEEE Trans. on Software Engineering*, vol. 16, no 2, Feb. 1990, pp. 141-152.

[22] B.Littlewood, J.L.Verrall, "A bayesian reliability growth model for computer software", *J. Royal Stat. Soc.*, (App. stat.), 22, 1973, pp. 332-336.

[23] T.J.McCabe, "A complexity measure", *IEEE Trans. on Software Engineering,* Vol. 2, no 4, Dec. 1976, pp. 308-320.

[24] J.McCall et al., *Methodology for software reliability prediction*, Rome Air Development Center, RADC-TR-no 87-171, vol. I, Nov. 1987.

[25] J.D.Musa, A.Iannino, K.Okumoto, *Software reliability: measurement, prediction, application*, McGraw-Hill, Singapore, 1987.

[26] P.D.T.O'Conner, *Practical reliability engineering*, Third edition, John Wiley & Sons, 1991.

[27] M.Paulk et al., "Capability maturity model for software", Version 1.1, CMU / SEI-93-TR-24, Feb. 1993.

[28] N.Ross, "The collection and use of data for monitoring software projects", *Measurement for software control and assurance*, Ed. B.A. Kitchenham et B. Littlewood, Elsevier Applied Science, London & New York, pp. 125-154.

[29] M.Xie, *Software reliability modeling*, Word Scientific, 1991.

[30] S.Yamada, M.Ohba, S.Osaki, "S-Shaped reliability growth modeling for software error detection", *IEEE Trans. on Reliability,* vol. R-32, no 5, 1983, pp. 475-478.

ACKNOWLEDGMENTS

The author gratefully acknowledges the conference organizers for giving her the opportunity to present the ideas developed in the paper. Special hanks are also given to Alain Costes, Mohamed Kaâniche and Jean-Claude Laprie for their invaluable comments while reading a draft version of this paper.

This work was partially supported by the ESPRIT Basic Research Action PDCS (Predictably Dependable Computing Systems, action no. 6362).

BIOGRAPHY

Karama Kanoun, Tel: (33) 61 33 62 35 Fax: (33) 61 33 64 11

Dr. Karama Kanoun is currently Chargé de Recherche at CNRS. She joined LAAS-CNRS in 1977 as a member of the "Fault-Tolerance and Dependable Computing" group. Her current research interests include modeling and evaluation of computer system dependability considering hardware as well as software. She received the Certified Engineer degree from National School of Civil Aviation, Toulouse, France in 1977, the Doctor-Engineer degree and the Doctor-ès-Science degree from the National Institute Polytechnique of Toulouse in 1980 and 1989 respectively. She is author of more than fifty publications in national and international journals and conferences. She has conducted several research contracts and she has been a consultant for several french companies and for the International Union of Telecommunications.

Dr. Kanoun is a member of the working group of the European Workshop on Industrial Computer Systems (EWICS): "Technical Committee 7 - Reliability, Safety and Security" and of the AFCET working group Dependability of Computing Systems. Likewise, she is a Program Committee member in several international conferences; she is serving as Program Committee co-chair of the Fifth "International Symposium on Software Reliability Engineering", ISSRE-94, and General Chair of ISSRE-95 to be held in Toulouse, in October 1995.