Reliability of a commercial telecommunications system

Mohamed Kaâniche, Karama Kanoun

Reliability of a Commercial Telecommunications System

Mohamed Kaâniche and Karama Kanoun

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

Abstract

We analyze data collected on a commercial telecommunications system and summarize some of the lessons learned from this study. The data correspond to failure and fault information recorded during system validation and operation: 3063 trouble reports corresponding to a five year period during which 5 versions of the system have been developed and more than one hundred systems have been introduced in the field. The failure information includes software failures as well as hardware failures due to design faults.

1. Introduction

The quality and dependability of a product can only be improved through a complete program that has to be followed from the earliest phases of the development process. A major part of a quality program is based on the analysis of the problems revealed in the considered product as well as in previous releases and similar products to make recommendations for future developments, i.e., learn from past experience. The results presented in this paper concern the analysis of failure and correction data collected on a commercial telecommunications system, including hardware and software failures. The system considered is a new generation of a family of products; the results presented constitute a part of more global results of a program aimed at improving the quality of this family. The data were collected during validation and operation phases.

Since quality programs are prerequisite to system quality, most of the companies have their own programs and collect and analyze their own data. Several papers have been devoted to the analysis of failure data within a quality program, they are either specific to telecommunications systems (see e.g., [1-8]) or related to more general application domains such as in [9, 10]. Moreover, most of the published results dealing with reliability growth mainly focus on software reliability. However, hardware reliability growth resulting from the removal of design faults has been seldom studied. The data analyzed in this paper cover both software and hardware reliability growth.

The paper is structured as follows. Section 2 presents the system and the data collected. In Section 3 preliminary analyses of collected data are performed whereas some statistics on the software failures and faults are given in Section 4. Section 5 gives the evolution of the failure intensities of the hardware, the software and its main functions. The failure rates are evaluated in Section 6.

2. System and data presentation

The system development started at the end of the eighties. A series of versions have been successively developed and released to the customers. Each new version includes a set of new features and improves the system by correcting the faults identified in previous versions. The development teams are organized into specification teams, design teams, coding teams and finally integration and system testing teams. This organization allows concurrent development of multiple versions: when the specification of version “i” is completed, specification documents are delivered to design teams, and the specification of version “i+1” starts. This process is also iterated for the other activities. Three stages can be distinguished in the life cycle of each version. The first stage corresponds to the traditional development phases: specification, design, coding and integration & testing. In the second stage, the version is delivered to a specific committee for acceptance testing, i.e. validation before release. Finally, the version is delivered to the customers. It is noteworthy that the previous products that belong to the same family of telecommunication systems were developed according to the traditional waterfall model. Therefore, the development of the system analyzed in this paper required a cultural and organizational evolution.

The study covers five versions (referred to as v.1 to v.5 in this paper): v.1 and v.2 are prototypes that were not delivered to the customers. These prototypes have been tested by the company, also beta-testing and acceptance testing have been applied to these versions. v.3 was the first commercial version. The system has been progressively introduced in the field; more than one hundred installations were operational at the end of the period considered.

As part of the product assurance quality activities, a large amount of data are collected reporting failure and correction information. This study covers a period of five years including validation and operation. Two periods can be distinguished: the first period covers the beginning of data collection until the release of v.3 and the second period starts from the v.3 release until the end of data collection. The data collected during the first period corres-

pond to the validation of v.1, v.2 and v.3. However, the second period includes data related to the operational use of v.3, v.4 and v.5 as well as data related to the validation of v.4 and v.5. This is due to the interleaving of development and operation phases in the development process.

2.1. Data collection

Failure reports (FRs) are used to record any discrepancy between the expected and the observed system behavior. The FRs received from the customers or from validation teams are recorded in a database and then analyzed by a specific Committee. Only one FR is kept per observed failure (i.e., rediscoveries are not recorded): if several FRs cover identical failures, only one (the first) is entered into the database. An FR is both a failure report and a correction report since it also contains information about the fault(s) that resulted in an abnormal behavior of the system. In our study, we used the following information extracted from the available FRs.

Registration date: gives the day during which the abnormal behavior was noticed.

Closure date: this information is not available on all FRs. The date indicated refers to the day when the solution is found and tested and not to the date of fault fixing in the operational sites. The following situations may occur:

- If the faults are identified and the corrections are implemented and tested then the FR can be closed.
- The analysis of the FR may lead to classifying it as irrelevant if it does not correspond to a failure of the system. Then, the closure date corresponds to the day when the FR is stored in the archive database.

Type: FR classification as hardware, software, documentation, irrelevant (duplicate FRs or reports eliminated because they do not refer to system failures for instance), others (firmware, errors in operating procedures, etc.).

Fault location: identifies the components in which the faults were corrected.

Version: gives the version in which the fault(s) corresponding to the failure was (were) corrected (not the version in which the failure occurred).

2.2. Software Description

The software is implemented according to a modular architecture. It is composed of 77 Atomic Components (ACs) fulfilling elementary functions. These ACs can be grouped into the following functions (this grouping resulted from a discussion with the system developer):

- Basic: low level operating system functions,
- Switching: switching and synchronization,
- Billing,
- Call processing,
- Support: user interface management, system support and supervision,
- Terminals: junctors and subscribers interfaces.

The number of ACs and the size in kilo-lines of source code (KLOC) of these functions are given in Table 1 for v.4 and v.5. The sum of ACs (79) is higher than the number of ACs of the software because two ACs appear in two different functions. Table 1 shows a large variation of the functions' size. There are two large functions (Support and Terminals), two others can be considered as medium (Billing and Call processing) and two small.

Function	# of ACs	Size v.4	Size v.5
Basic	12	33	32
Switching	9	35	40
Billing	6	87	80
Call processing	11	107	113
Support	24	272	257
Terminals	17	304	294
Total	79	836	815

Table 1 - Number of ACs and size in KLOC of software functions.

3. Preliminary analyses

The database contains 3063 FRs collected during five years. Table 2 presents the number of FRs by type. The software FRs constitute the most important part of the data. Even though the hardware and documentation FRs relative percentages are not high, the corresponding number of FRs is significant. It is noteworthy that only hardware design faults are included in the database. Usually, these data are ignored and only physical faults are taken into account in the analysis of hardware reliability.

Type	#FRs (%)	
Software	1853	(60.5%)
Hardware	195	(6.4%)
Documentation	165	(5.4%)
Irrelevant[1]	716	(23.4%)
Others	134	(4.4%)
Total	3063	

Table 2 - Number of FRs by type

The analysis of time evolution between registration and closure of FRs gives insights into the maintenance effort needed to solve the problems met by customers during operation for instance, and to follow up the efficiency of the FR analysis process. This analysis requires the identification of FRs with closure dates. Only 2446 FRs have closure dates. Regarding the non closed 617 FRs, three possibilities may hold: a) these FRs have not been solved yet or b) they have been solved but the closure dates are not recorded and c) even if a solution is found, the development team decided to integrate the corrections later.

Figure 1 plots the number of FRs recorded and the number of closed FRs per unit of time (UT). Both curves have similar shapes but they are skewed in time due to the correction delay: an increase of the recorded FRs is followed by an increase of the closed FRs later. Figure 2 plots the FRs life duration, i.e. the delay between the registration and the closure of FRs. It appears that most

[1] This percentage is not very high when compared to those observed for other real-life systems: for example the figures published in [7, 11, 12] show that about half of the FRs are irrelevant.

FRs are solved in few months. However, about 12% have a life duration higher than 12 months (the FRs without indication of the closure date are included in this set).

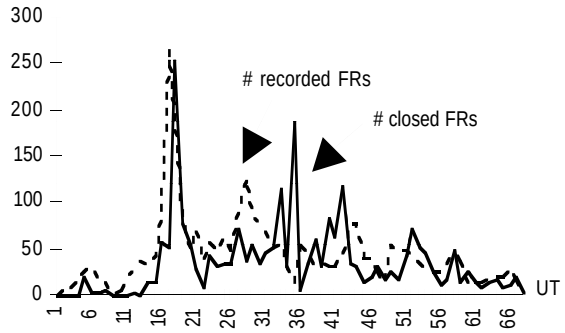

Figure 1 - Evolution of the number of recorded FRs and closed FRs

Figure 2- Satistics on FRs life duration

4. Software FRs: descriptive analyses

Due to space limitation, we limit the analyses to: a) the relationship between FRs and corrected faults, b) the distribution of faults and fault density and c) the fault distribution among the most error prone function ACs.

4.1. Software FRs and corrected faults

Analysis of the relationship between failures and corrected faults can be based only on those FRs for which the fault location is indicated. Among the 1853 software FRs, only 1512 identify the corrected ACs: they are denoted CFRs. Since some failures led to corrections in more than one AC, the number of corrected faults in ACs obtained from the analysis of the 1512 CFRs and the associated ACs is equal to 1801. Table 3 shows the number of CFRs that led to the modification of 1, 2, etc. ACs. The high percentage of CFRs that led to modify only one AC denotes a strong independence between ACs. A large proportion of the CFRs that led to modify more than one AC is attributed to "Terminals" which contains some highly dependent ACs implementing different types of junctors.

The fault reduction factor defined as the ratio between the number of failures and the number of corrected ACs estimated from Table 3 is around 0.91. This value is of the same order of magnitude of fault reduction factors estimated in other case studies (see for instance [13])

# ACs modified	# CFRs (%)	
1	1348	(89%)
2	103	(7%)
3	32	(2%)
≥ 4	229	(1.8%)

Table 3 - CFRs and ACs modified

4.2. Fault distribution and fault density analysis

Identifying the most error prone functions is helpful for debugging and maintenance tasks. As in Section 4.1, the analysis is based on the set of 1512 CFRs. A quick analysis shows that 38% of ACs contain 80% of all the software corrected faults (the 38% of ACs constitute about 54% of the code). The distribution of faults in the functions is presented in figure 3. As expected, the relative distribution of faults is consistent with the relative sizes of the functions. "Support" and "Terminals" correspond to 67% of the software size and contain 71% of the corrected faults. Compared to the previous products developed by the same company, these functions are also those supporting more new features than the others, due to the large increase of the system processing capacity and also the introduction of sophisticated Human Computer Interfaces. Therefore, during the development of these functions the developers have faced new problems. Hence, the knowledge cumulated through feedback from previous products could not be used to reduce the number of faults in these functions.

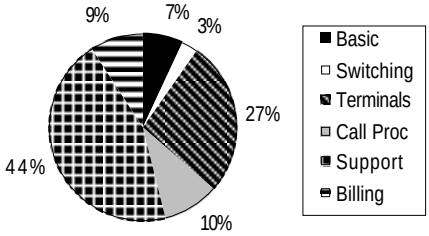

Figure 3 - Distribution of faults in respective functions

Fault density, defined as the number of faults divided by size, is a common measure generally used to evaluate software code quality. Figure 4 plots the fault density for software functions: "Support" has the highest fault density while similar results are obtained for the other functions.

Figure 5 plots the fault density distribution among ACs: similar distributions are obtained before and after v.3 release, however the proportion of ACs having low fault density is higher for the period after v.3 release (even though the corresponding period is longer): about 95% of ACs have a fault density less than 3 faults/KLOC. This result is comparable to fault density values reported for other systems [14, 15]. It can be concluded that the quality of code is on average similar to what is commonly achieved in the field. It is noteworthy that the fault densities reported correspond to five years of data collection, which is a long period compared to the studies reported which are generally limited to the first year of operation.

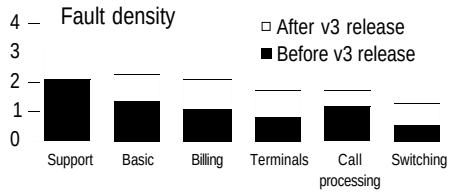

Figure 4 - Fault density distribution among software functions

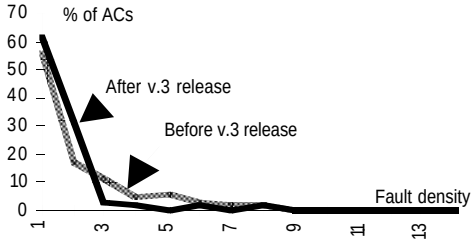

Figure 5 - Fault density distribution among ACs

4.3. Faults distribution in "Support" function

Function "Support" deserves particular analyses as it is the one most error prone. The ACs belonging to this function can be further grouped into three sub-functions: **HI** (human-system interaction), **SV** (system supervision) and **SP** (other system support functions).

These sub-functions respectively account for 34%, 24% and 42 % of the function "Support" total size. Detailed statistics on the distribution of corrected faults among HI, SV and SP sub-functions are given in Table 4.

Sub-function	**HI**	**SV**	**SP**
Corrected faults	16 %	40 %	44 %
Before v.3 release	7 %	50 %	43 %
After v.3 release	31 %	22 %	47 %

Table 4 - Corrected faults distribution among H, SV and SP

It appears that most of the faults discovered were contained in SV and SP. The combined analysis of the relative distribution of size and corrected faults shows that SP is the most error prone part. Generally, the design and validation of supervision functions are complex because this part of the system mainly deals with the abnormal behavior of the system that is difficult to specify. Moreover, the analysis of fault distribution, before and after v.3 release, shows that HI and SV exhibit a particular behavior: most of the HI faults were revealed after the system delivery, whereas the opposite situation is observed for SV. This is confirmed by Figure 6, which indicates for each AC the number of faults corrected before and after the release of v.3 and by Figure 7 which gives the evolution of the cumulative number of corrected faults in each sub-function. The identification of faults in HI functions is facilitated by the use of the system in the field. This is a more general result that is due to the fact that end-user requirements related to system exploitation and maintenance are difficult to identify early in the development process. More involvement of these users in the definition of system requirements and extensive use of prototyping techniques should lead to the early detection of HI faults.

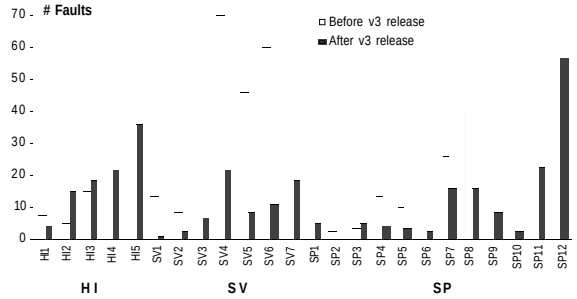

Figure 6- Corrected faults distribution for each AC of HI, SV and SP

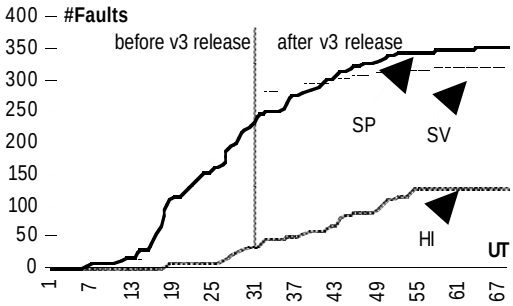

Figure 7 - Cumulated number of corrected faults in HI, SV and SP

5. Failure intensity

Failure intensity, characterizing the number of failures per unit of time, is a good indicator of the reliability evolution. First, we analyze the evolution of the number of software and hardware failures reported from all system installations. In a second step, we show the relationship between the failures and the versions of the software. Then we consider the failure intensity of an average system.

5.1. Hardware and software failure intensities

Figure 8 plots the evolution of the software failure intensity. We can notice a sharp increase of failure intensity around UT 17. Before that date, the failure intensity has been globally increasing. Afterwards, it was decreasing denoting a global reliability growth. However, local variations in failure intensity are observed due to a) the introduction of new versions (see Section 5.2) and b) the gradual installation of new systems (see Section 5.3).

Figure 9 plots the hardware failure intensity evolution (195 failures due to design faults have been recorded). A large proportion (40%) appeared during the first 15 UTs. During this period, the hardware failure intensity is about 5 failures/unit of time compared to about 20 failures/unit of time for the software while the hardware FRs constitute only 7% of the relevant data (See Table 2). This result is not surprising because the hardware design faults are often revealed at the beginning of the testing period.

5.2. Software FRs distribution among versions

Figure 10 plots the software failure intensity evolution per version. Even though all the software FRs are not

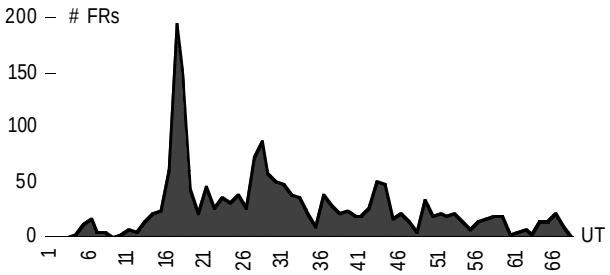

Figure 8 - Software failure intensity

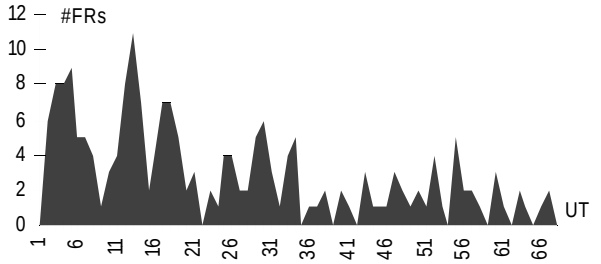

Figure 9- Hardware failure intensity

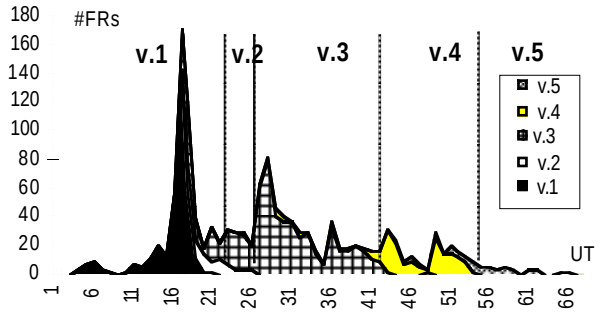

Figure 10 - Evolution of Software FRS with the versions

considered here (not all of them indicate the corrected ACs), this curve is very similar to that of figure 8. The approximate periods of validation of each version are indicated in figure 10. We can clearly see that some of the FRs corresponding to the peak at UT 17 remained not corrected until v.3: v.2 was just a slight modification of v.1, with addition of some features. It was under development for a very short period and a large part of FRs in v.2 were deferred until the development of v.3. The validation of v.3 started around UT 26 and its commercialization around UT 31. Note that the failures occurring before UT 26 and attributed to v.3 have been observed in v.1, v.2 or v.3 while the corresponding corrections have been introduced in v.3.

Considering figure 8 and figure 10 together, one can see that the peaks observed in figure 8 around UT 27 and UT 43 are directly linked with the beginning of v.3 and v.4 validation whereas the peak observed around UT 36 corresponds to the beginning of the v.3 operational life. It is noteworthy that similar curves have been reported in [4].

5.3. Average system software failure intensity

In order to analyze software reliability as perceived in time by the users, failure intensity has to be normalized by considering that of an average system (i.e., failure intensity divided by the number of systems in use). Figure 11 plots the failure intensity for an average system during the operation period starting at the release of v.3. Due to the high number of systems in operation, the average failure intensity observed during the operation period is some orders of magnitude lower than the failure intensity observed before the release of the first commercial version to the field. This illustrates the software reliability growth.

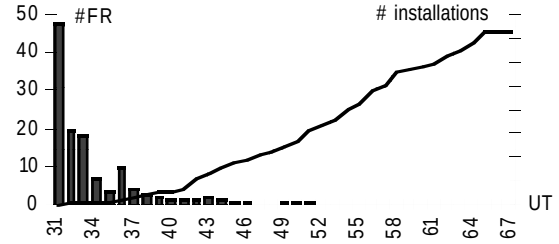

Figure 11 - Software failure intensity for an average system and evolution of the number of installations

6. Reliability evaluation

This section is devoted to reliability growth model application to estimate the software and hardware failure rates in operation. Models permit predictions of the product reliability which enable proper planning of the maintenance effort. This, in turn, minimizes the maintenance cost without decreasing customer satisfaction. Reliability predictions give the developer more knowledge about the product and the corresponding development process and could be helpful for the development and maintenance of further products. Comparison of the modeled product with the "state of the art" can provide information about possible optimization of the development process. Moreover, the evaluation of software functions reliability leads to the identification of the system parts that need special attention during the development and validation phases.

For this purpose, we have applied several reliability growth models to the observed failure intensity of an average system, in particular the hyperexponential model (HE) [16]. HE is a non homogeneous Poisson process model characterized by the following failure intensity [16] (α, λ_{sup} and λ_{inf} are to be estimated from the data):

$$h(t) = \frac{\alpha \lambda_{sup} e^{-\lambda_{sup} t} + (1-\alpha) \lambda_{inf} e^{-\lambda_{inf} t}}{\alpha e^{-\lambda_{sup} t} + (1-\alpha) e^{-\lambda_{inf} t}}, \quad 0 \le \alpha \le 1, \ \alpha + (1-\alpha) = 1$$

h(t) is non-increasing with time from $h(0) = \alpha\lambda_{sup} + (1-\alpha)\lambda_{inf}$ to $h(\infty) = \lambda_{inf}$: λ_{inf} is referred to as the residual failure rate. This model has been applied with success to several data sets (see e.g. [3]).

We will first apply the HE model to the whole software, then to the software functions and finally to the hardware system. We concentrate on the operational period starting at v.3 release, considering an average system.

The result of the HE model application to software data is given in figure 12. The residual failure rate for this case equals $1.4\ 10^{-3}$ /h. The residual failure rates estimated by

the HE model for the various functions are given in Table 5. The highest value corresponds to the "Support" function which contains the highest number of CFRs.

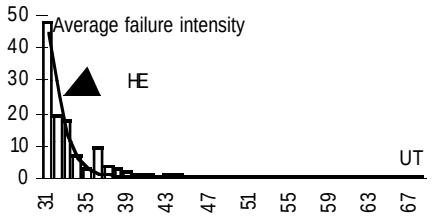

Figure 12 - HE model applied to the software failure data

Function	Residual failure rate (/h)
Basic	$8.2\ 10^{-6}$
Switching	$3.0\ 10^{-6}$
Billing	$2.7\ 10^{-5}$
Call processing	$4.3\ 10^{-5}$
Support	$2.7\ 10^{-4}$
Terminals	$7.7\ 10^{-5}$

Table 5 - Software functions residual failure rates (failures/hour)

Figure 13 plots the HE model applied to the hardware failure intensities for an average system. The residual failure rate is $2.5\ 10^{-5}$ /h. The estimated hardware failure rate is thus about 50 times smaller than the estimated software failure rate. However, we should keep in mind that only design hardware faults are included in this evaluation. One should add the failure rate due to physical faults to obtain the failure rate of the hardware.

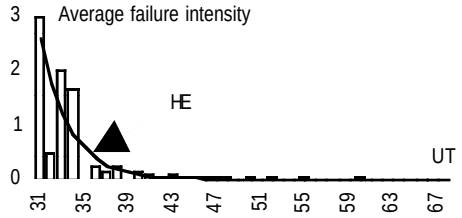

Figure 13- HE model applied to the hardware failure data

7. Conclusion

The data reported in this paper allowed us to analyse several issues related to the quality and the reliability of a telecommunications commercial system. Both software failures and hardware failures due to design faults are considered. Regarding the results obtained for the software, the estimated fault densities as well as the failure rates are of the same order of magnitude of what we have already observed on other similar telecommunications systems. Therefore, the organizational and cultural evolution due to the modification of the development process did not have a negative impact on the reliability of the system. Most of the problems discovered in the software were attributed to new functionalities and major modifications of the previous products belonging to the same family. Unfortunately, the data analyzed in this paper did not allow us to explicitly analyse the impact of reuse on software reliability and quality. The enrichment of the data collection procedure by reporting this kind of information is necessary to make these analyses. The other interesting aspect of the data analysed in this paper concerns hardware failures related to design faults. Usually, these failures are not accounted for in systems reliability evaluation. However, given the relative importance of these failures, hardware design faults should no longer be ignored. These failures can be analysed using all the body of the results that have already been applied for software reliability growth evaluation.

Acknowledgment

The authors are indebted to the company which provided the data analysed in this paper and, especially, to the engineers who participated to the analysis of these data.

References

[1] W. D. Jones, "Reliability Modeling for Very Large Soft-ware Systems in Industry," 2nd Int. Symp. on Software Reliability Eng., Austin, USA, 1991, pp. 35-42.

[2] K. Kanoun and T. Sabourin, "Software Dependability of a Telephone Switching System," 17th Int Symp. on Fault-Tolerant Comp., Pittsburgh, USA, 1987, pp. 236-241.

[3] K. Kanoun, M. Kaâniche, and J.-C. Laprie, "Experience in Software Reliability: From Data Collection to Quantitative Evaluation," 4th Int. Symp. on Software Reliability Eng., Denver, CO, USA, 1993, pp. 234-245.

[4] G. Q. Kenney and M. A. Vouk, "Measuring the Field Quality of Wide-Distribution Commercial Software," 3rd Int. Symp. on Software Reliability Engineering, Research Triangle Park, NC, USA, 1992, pp. 351-357.

[5] D. W. Carman, A. A. Dolinsky, M. R. Lyu, and J. S. Yu, "Software Reliability Engineering Study of a Large-Scale Telecommunications System," 6th Int. Symp. on Soft. Rel. Eng., Toulouse, France, 1995, pp. 350-359.

[6] W. K. Ehrlich, J. P. Stampfel, and J. R. Wui, "Application of Software Reliability Modeling to Product Quality and Test Process," Int. Conf. on Software Eng., Nice, France, 1990, pp. 108-116.

[7] Y. Levendel, "Reliability Analysis of Large Software Systems: Defects Data Modeling," *IEEE Trans. Software Engineering*, vol. SE-16 (2), pp. 141-152, 1990.

[8] K. Kaâniche, K. Kanoun, M. Cukier, and M. Bastos Martini, "Software Reliability Analysis of Three Successive Generations of a Switching System," *First European Conference on Dependable Computing*, Berlin, Germany, 1994, pp. 473-490.

[9] R. Chillarege and S. Biyani, "Identifying Risk Using ODC Based Growth Models," 5th Int. Symp. on Software Reliability Eng., Monterey, USA, 1994, pp. 282-288.

[10] M. K. Daskalantonakis, "A Practical View of Software Measurement and Implementation Experiences Within Motorola," *IEEE Trans. on Software Engineering*, vol. SE-18 (11), pp. 998-1010, 1992.

[11] V. R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans.on Software Eng.*, vol. 10, pp. 728-738, 1984.

[12] M. Kaâniche, K. Kanoun, and S. Metge, "Failure Analysis and Validation Monitoring of a Telecom-munication Equipment Software System," *Annales des Telecom.*, vol. 45, pp. 657-670, 1990, *(In French)*.

[13] J. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. McGraw-Hill, 1987.

[14] R. Gibson, *Managing Computer Projects*. London, UK: Prentice Hall, 1992.

[15] M. Dyer, *The Cleanroom Approach to Quality Software Development*. John Wiley & Sons, Inc., 1992.

[16] J.-C. Laprie, K. Kanoun, C. Béounes, and M. Kaâniche, "The KAT (Knowledge-Action-Transformation) Approach to the Modeling and Evaluation of Reliability and Availability Growth," *IEEE Trans. Software Engineering*, vol. SE-17, pp. 370-382, 1991.