Qualitative and Quantitative Reliability Assessment

KARAMA KANOUN, MOHAMED KAÂNICHE, and JEAN-CLAUDE LAPRIE
French National Organization for Scientific Research

Traditional system reliability efforts have failed to focus on software reliability. To remedy this, the authors propose a method that uses descriptive analyses, trend analyses, and reliability models to control testing activities, evaluate software reliability, and plan maintenance.

During the development of a software system, the supplier must efficiently monitor the development activities; comply with the delivery schedule; predict when a specified level of reliability will be reached, or at least check how well the software will satisfy the customer's requirements; and reduce maintenance efforts. On the other hand, the customer needs a reliable product, delivered on time and at the lowest price. Our method helps reach these goals. It is based on the analysis and evaluation of software reliability by processing failure data collected on a software product during its development and operation.

Traditionally, system reliability efforts have not focused on software reliability. We believe that failure prediction can be improved if software reliability modeling is integrated into an overall approach. The method we propose is based on the combined use of descriptive analyses, trend analyses, and reliability models to control testing activities, evaluate software reliability, and plan maintenance.

0740-7459/97/$10.00 © 1997 IEEE

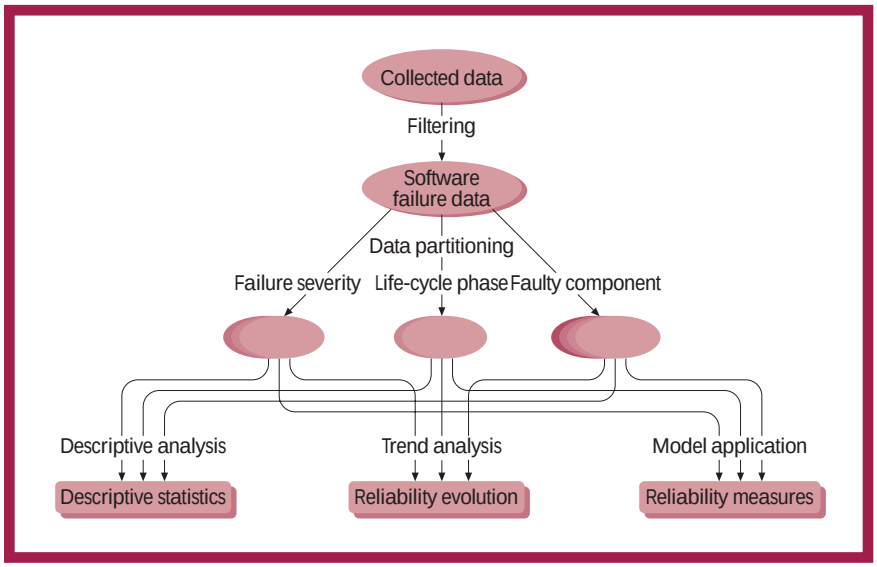

Figure 1. *The various steps of reliability analysis and evaluation.*

Even though each reliability evaluation can be considered a special case due to the diversity of several factors, including

- the software's nature and the corresponding failure data,
- the development and validation methods adopted,
- the organization of data collection, and
- the aims of the analysis,

our experience let us define a general method for software reliability analysis that covers the entire life cycle, from development to operational use. This method emphasizes real-time data processing within a controlled environment that allows efficient, real-time feedback. Doing so helps control the development process and quantify software reliability. To facilitate the processing of failure data, we developed the Sorel tool, which allows automatic data processing. The box on page 79 gives a more detailed description of Sorel.

OVERVIEW

The objectives of a reliability study are directly related to the point of view considered—whether supplier or customer—and the life-cycle phase concerned. Generally, the main objectives of development—follow-up, maintenance planning, and reliability evaluation—are expressed as measures.

During software development, important measures to track include

- the evolution of the trend—whether reliability increases or decreases—in response to the testing and debugging effort to monitor these activities, and
- the number of failures expected to occur over several subsequent time periods so as to plan the test effort and thus the testing time and the size of the test team.

Once the software is implemented and operating, two types of measures become useful.

- From the customer's perspective, the failure intensity, mean time to failure, or failure rate are important because they help evaluate whole-system reliability, including hardware and software.
- From the supplier's perspective, the expected number of failures among all installations, or the number of corrections to be performed, are important because they help estimate the maintenance effort still needed.

These measures determine the nature of data to be collected and the kind of data processing to be done. Two categories of data can be recorded:

- data characterizing the product itself, the production process, and the use environment, such as the software size, language, functions, current version, verification and validation methods, tools used, workload, and so on; and
- data associated with failures and corrections, such as date of occurrence, nature of failures, consequences, fault types, fault location, and so on.

Usually, data is collected through use of failure and correction reports. Figure 1 summarizes the various operations you can perform on the collected data set and the results you may expect from these operations, which include descriptive statistics, reliability evolution, and reliability measures. The collected data may include foreign data, which requires filtering to keep only the data related to software. Depending on your study's objectives, you may perform reliability analysis using the whole data set or subsets of it obtained through data partitioning. The latter approach enables more detailed analyses yielding more elaborate results.

Three types of analysis may be performed on the whole data set and on the derived subsets.

- Descriptive analyses are based on statistics that address fault density, the number of faults per release, or combined analyses such as fault location and failure severity. They are not directly related to software reliability evaluation; they enhance knowledge about the software and the corresponding failure data.
- Trend analyses concern the time evolution of reliability measures, such as the time to failure or the number of failures, which help gauge the effectiveness of the testing activities to be assessed. Trend analyses also lead to better estimations when using reliabili-

ty growth models, because those models can be applied directly to subsets of the data that display trends that confirm the underlying assumptions, rather than blindly.

♦ Application of one or several reliability growth models lets you predict the reliability measures based on the observed behavior of the software.

Trend analysis and reliability evaluation are performed on data in the form of time to failure or number of failures per unit of time. The latter are also called grouped data. These values are extracted from the failure and correction reports. The choice between the two forms is governed by the objective of the study and the life cycle phase being analyzed. Grouped data can be collected easily and helps mitigate the impact of local fluctuations on software reliability evaluation. The unit of time used for grouped data is a function of the system usage type and the number of failures occurring during the period analyzed, and may differ for different phases.

FILTERING, PARTITIONING, ANALYSIS

Operations on data filtering, partitioning, and descriptive analysis are specific to the system under study, the way data has been collected, the purpose and structure of the software, and the objectives of the analysis.

You may not need all these successive operations in every reliability study. For example, if you enter the collected data into a database and check it on entry, filtering is not needed. You would perform data partitioning and analysis depending on the level of detail sought.

Collected data may include, in addition to the reports related to actual software failures, extraneous data such as false trouble reports or duplicate data, which must be discarded.

SOREL

Sorel is a software reliability analysis and evaluation tool composed of two modules, for testing trends and modeling the growth of reliability (see Figure A). It can operate on two types of failure data: interfailure times and number of failures per time unit. The two modules operate on the same input data files, which can be created and changed by any word processing or graphics editor. Numerical results are displayed automatically on the screen during execution: the user can also ask for the corresponding curves. The results are recorded as ASCII files that can serve as input to other applications, allowing for instance comparison of results issued from different models.

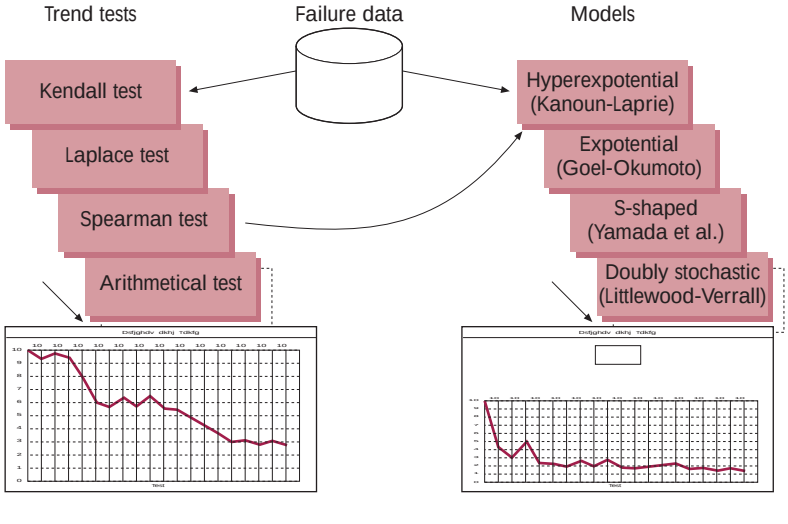

Figure A. *Sorel overview.*

Sorel runs on the Macintosh II-*xx* with an arithmetical coprocessor. We gave the interface special attention, making it interactive, menu-driven, and ensuring that it takes advantage of the Macintosh's multiple-window management facilities. Sorel is modular, so new reliability growth tests and models can be added easily. Written in 5,000 lines of Pascal, it requires about 300 Kbytes of memory.

REFERENCE

1. K. Kanoun et al., "Sorel: A Tool for Reliability Growth Analysis and Prediction from Statistical Failure Data," *23rd Int'l Symp. Fault-Tolerant Computing*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 654-659.

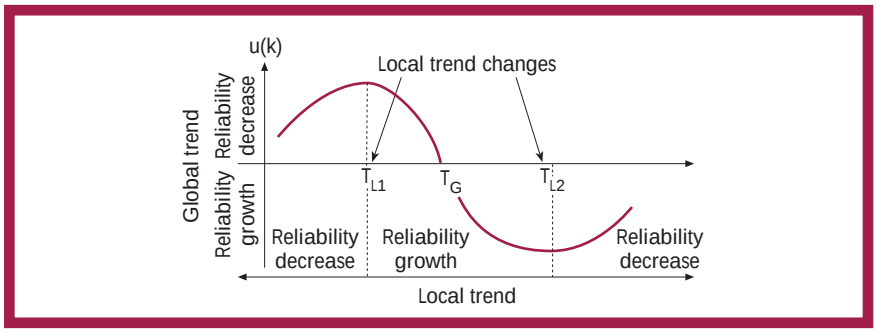

Figure 2. *Laplace factor and trend changes.*

Rediscoveries—such as multiple activation of the same fault—must be kept when adopting the customer's viewpoint, because they correspond to different failures occurring on the same or a different installation and as such affect observed reliability. Data filtering—also known as data validation—though time-consuming and cumbersome, accomplishes this. It requires precise knowledge of the system and software and you must interview those involved in software testing and data collection. When you enter the trouble reports into a database, you can partly automate the filtering. From our experience[1] and that of others,[2,3] we have found that about half the failure reports generated by a supposed software failure are either unusable or, after a detailed examination, reveal problems that cannot be attributed to software.[4] Thus, data filtering is essential because accurate reliability analysis can only be conducted on filtered data.

You must partition your data into subsets whenever you require a detailed analysis. The most common partitions concern failure severity and faulty software components.

Applying reliability growth models to the most severe failures allows for example evaluation of the software failure rate corresponding to the most critical behavior. This failure rate is generally more significant than that for the whole software product, which may also incorporate failures with no major impact on the system's behavior. Partitioning according to fault location and evaluation of the failure rate of various components allows measurement of each component's influence on the whole failure rate and identification of the most and least reliable components.

Descriptive analysis relies on synthesis of the observed phenomena in the form of control charts or tables that identify the most significant phenomena. The description may consist of simple analyses such as fault density, fault typology, and fault distribution among new, modified, and reused software components. Alternatively, it could derive from a combined analysis, such as the relationship between failure occurrence conditions and criticality; fault location and failure criticality; or the number of faults in the components and the component size. Such statistics are commonly used by some companies.[4,5] The accumulation and analysis of information about several projects, products, and releases provides a company with better insights into its products and the impact of the development process on them.

TREND ANALYSIS

You can analyze reliability evolution with trend indicators. Three simple graphical tests can help determine whether the system becomes more or less reliable:

- time to failure,
- the cumulative number of failures, and
- the number of failures per unit of time (failure intensity).

You derive the trend empirically, using eyeball analysis, from the evolution of the plotted measures. While these plots provide a quick and useful indication of the trend, they may be ambiguous and even misleading in some situations because they fail to offer quantifiable means. Therefore, you need formal statistical tests to enhance confidence in the results. Such tests allow for trend quantification. Various statistical tests such as the Laplace, Kendall, and Spearman tests are available for identifying trends in time-series or grouped data. Previous studies have shown the Laplace test to be optimal within the framework of the most famous software reliability models.[6] This test consists of calculating the Laplace factor, u(T), for the observation period [0, T]. When expressed in terms of time to failure, the Laplace factor is

$$u(T) = \frac{\frac{1}{N(T)}\sum_{i=1}^{N(T)}\sum_{j=1}^{i}\theta_j - \frac{T}{2}}{T\sqrt{\frac{1}{12\,N(T)}}}$$

where θ_j is the time to failure j counted from system restart after failure $j-1$, and N[T] is the number of failures in [0,T]. In terms of $n(i)$, the number of failures during unit of time i, the expression of the Laplace factor is

$$u(k) = \frac{\sum_{i=1}^{k}(i-1)n(i) - \frac{(k-1)}{2}\sum_{i=1}^{k}n(i)}{\sqrt{\frac{k^2-1}{12}\sum_{i=1}^{k}n(i)}}$$

Significance levels are associated with

these statistics. For example, for a significance level of 5 percent, values of u(k) such as $-1.96 \leq u(k) \leq +1.96$ indicate stable reliability over [1, k]. In practice, in the context of reliability growth,

- negative values indicate a decreasing failure intensity and thus a reliability increase,
- positive values suggest an increasing failure intensity and thus a reliability decrease, and
- values oscillating between –2 and +2 indicate stable reliability.

In its classical form, the Laplace test gives the trend over a given interval of time—the global trend—and identifies the regions of global trend change as indicated by point T_G of Figure 2. The work we have performed[7] allows extension of the Laplace factor to detect the regions of local trend changes as indicated by points T_{L1} and T_{L2} of Figure 2. If you change the origin of the considered interval, and let the observation interval start at T_{L1}, u(k) becomes negative over the whole (remaining) interval. The change in the time origin does not result in a simple translation: removal of failure data from 0 to T_{L1} underlines the local variations and thus amplifies the Laplace factor variation. However, the points of local trend change are preserved.

From a pragmatic viewpoint, using the Laplace factor as a trend indicator, the procedure we've described lets global and local trends be defined as follows:

- negative or positive Laplace factor values over an interval indicate a global reliability increase or decrease, respectively, for that interval; and
- decreasing or increasing values of the Laplace factor over a subinterval indicate local reliability increase or decrease, respectively, for that subinterval.

In real situations, we use the Laplace factor to analyze the trend, considering its sign (plus or minus) along with its evolution. Doing so lets both global and local trends be identified "at a glance."

Using trend analysis results. Trend analysis is intended only to draw attention to problems that might otherwise pass unnoticed until too late, thus providing an early warning that will likely shorten the search for a solution. It can be used to enrich the interpretation of someone who knows the software from which the data is derived, the development process, and the user environment. Typically, we analyze three trends: decreasing reliability, increasing reliability, and stable reliability.

Decreasing reliability is generally expected and considered normal at the start of a new activity, such as a new life cycle phase, changing test sets within the same phase, adding new users, or activating the system with a different user profile. Decreasing reliability may also result from regression faults. Trend analysis reveals this kind of behavior. If the duration of the decrease seems long, you must pay attention to it. Those situations in which reliability continues to decrease can point to problems in the software: analyzing the reasons for the decrease and the nature of the activated faults is of prime importance in such a situation. This analysis may influence the decision to re-examine the corresponding software part.

Reliability growth that follows a reliability decline is usually welcome because it indicates that, after removal of the first faults, the corresponding activity reveals fewer and fewer faults. When calendar time is used, sudden reliability growth may result from a period during which the system is used less or not at all; it may also be caused by unrecorded failures. When you notice a reliability growth trend, you must take particular care and analyze the reasons for the sudden increase.

Stable reliability indicates that either the software is not receiving corrective maintenance, or the corrective actions have no visible effect on reliability. When the software is being validated, stable reliability with almost no failures means that the corresponding activity has reached "saturation": the application of the corresponding test sets reveals no new faults. At this point, you must either stop testing and introduce new test sets or proceed to the next phase. More generally, we recommend that you continue a test phase as long as reliability keeps growing and end it only when you reach stable reliability with almost no failures. Thus, in practice, if you have not reached stable reliability, your validation team (and its manager) may decide to continue testing before software delivery because it will be more efficient and cost-effective to remove faults during validation than during operation.

For the debugger, however, identifying the region in which the local trend changes is of prime importance: these regions herald the beginning of new trend periods. If the debugger enters a period of reliability decrease, he must be vigilant; he need not wait for the moment when the global trend changes to undertake corrective action. Conversely, if he enters a period of reliability growth, he becomes confident earlier.

As stated earlier, it is more efficient to exploit the results in combination with other criteria such as test coverage and development activities, and with information emerging from the descriptive analyses such as the nature of activated faults, failure consequences, or affected components. An example of applying trend analysis to monitor the development of a real-world software product can be found elsewhere.[4]

Trend analysis only draws attention to problems that might otherwise pass unnoticed.

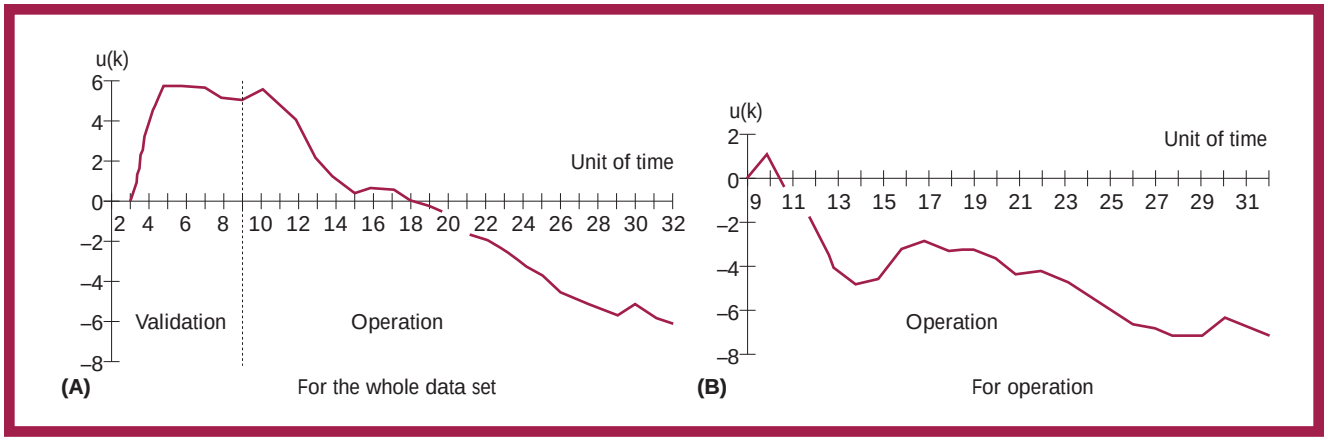

Figure 3. *Laplace factors for all the installed systems in operation: (A) for the whole data set; (B) only during the operational phase.*

TABLE 1
COMPONENT SIZE AND NUMBER OF CORRECTIONS IN OPERATION

Application	Size (Kbytes)	Number of corrections
Telephony	75	40
Defense	103	47
Interface	115	41
Management	42	18
Total	335	146

MODEL APPLICATION

Trend tests give information about trend evolution. However, if you want to evaluate the reliability of a particular piece of real-world software, you should apply reliability growth models. This lets you do the evaluation during validation. You are thus able to handle the delays and ensure that the software meets its reliability requirements. Growth models also let you assess the software's reliability in operation and thus estimate overall system dependability. Estimating the number of failures that will occur over a future time period is useful when planning the software maintenance effort. During development, frequent changes in the use environment restrict you to very short-term estimations. The relevance of the measures obtained from applying a reliability growth model varies with the considered life-cycle phase:

♦ Applying reliability growth models during the early stages of validation is not convincing—when the observed times to failure are of the order of magnitude of minutes or hours, the mean time to failure predictions obtained from such data can hardly exceed minutes or hours, which is distant from any expected reasonable reliability. In this case, software validation should be guided by trend analyses, as shown in the previous section.

♦ When the software being validated becomes more reliable, the time to failure may be large and the application of reliability growth models is more convincing, particularly when the software is activated under an operational profile.

♦ When the software is in operation, on multiple installations, the results are usually highly relevant since you have a larger sample of failure data.[8,9]

Models and trend analysis. Blindly applying a reliability growth model may lead to nonrealistic results when the trend displayed by the data differs from the one assumed by the model. However, if the model is applied to data displaying a trend that behaves according to its assumptions, results may improve dramatically.[9,10] This is because the already existing reliability growth models only allow two types of behavior to be modeled: decreasing failure intensity, or increasing failure intensity prior to undergoing decreasing failure intensity. Thanks to trend analyses, failure data can be partitioned according to trend, and reliability growth models can be selected as follows:

♦ In the case of reliability growth, most existing reliability growth models can be applied.

♦ When the failure data exhibits decreasing reliability followed by reliability growth, we recommend an S-shaped model.

♦ When the system displays stable reliability, you can apply a constant failure intensity model; reliability growth models are not needed in this case.

Models in real time. To predict the software's future behavior based on the information available at a given time, you must carry out a trend test on the available data. This helps you choose the reliability growth model or models to be applied and the subset of data to which they should be applied. The models are applied as long as the environmental conditions remain significantly unchanged, showing no major changes in the testing strategy or specifications and no new system installation with different operational profiles. Even in these situations, declining reliability may be noticed. Initially, you can consider this to result from a local, random fluctuation and assume that reliability will increase sometime in the near future. Thus, you can still rely on the predictions without partitioning data. If reliability continues to decline, you must find the reasons why. Meanwhile, new predictions may be made by partitioning data into subsets according to the new trend displayed by the data.

If a significant change in the devel-

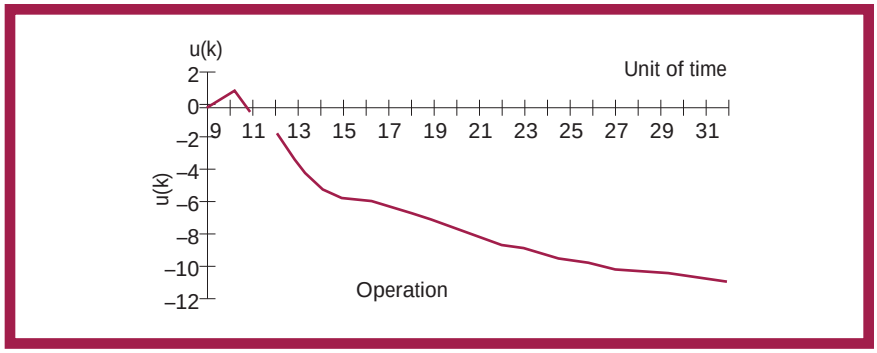

Figure 4. *Laplace factors for an average system produce a smooth curve, in contrast to Figure 3.*

opment or operational conditions occurs, you must be aware that local reliability trend changes may result, leading to erroneous predictions. If there is insufficient evidence that a different phase in the program's reliability evolution has been reached, you can trust the results of reliability growth models. If there is an obvious reliability decrease, you must wait until a new reliability growth period is reached, at which time the data must be partitioned according to the new trend.

We developed Sorel to make it easier to process failure data and to apply trend analysis and growth models in combination.

CASE STUDY

To explore the application of our Sorel tool, we consider the failure data collected on an electronic switching system (ESS) that was installed at 42 different sites during the collection period. The software faults detected and removed were recorded in appropriate failure reports, known as FRs. Raw data were filtered before being entered into the database: the 210 FRs correspond to genuine software faults removed and recorded over 32 time units, including the end of validation (8 time units, 73 FRs) and the beginning of operation (24 time units, 137 FRs). The ESS software consists of four components corresponding to the system's four main functions: telephony (all modules that provide switching), defense (all online testing and reconfiguration mechanisms), interfaces (all those with local devices, including memories, terminals, alarms, and so on), and management (programs that allow communication with external devices). Table 1 gives the size of the various components and the number of corrections carried out on each during the system's operational life. The sum of the corrections made on different components during operation, 146, is higher than the number of corrections performed on the software as a whole during operation, 137. This is because 11 failures led to the modification of more than one component.

To plan for the ongoing maintenance effort, we analyzed the trend, evaluated the operational reliability for the whole software and its various components, then estimated the number of failures that would occur over a future period.

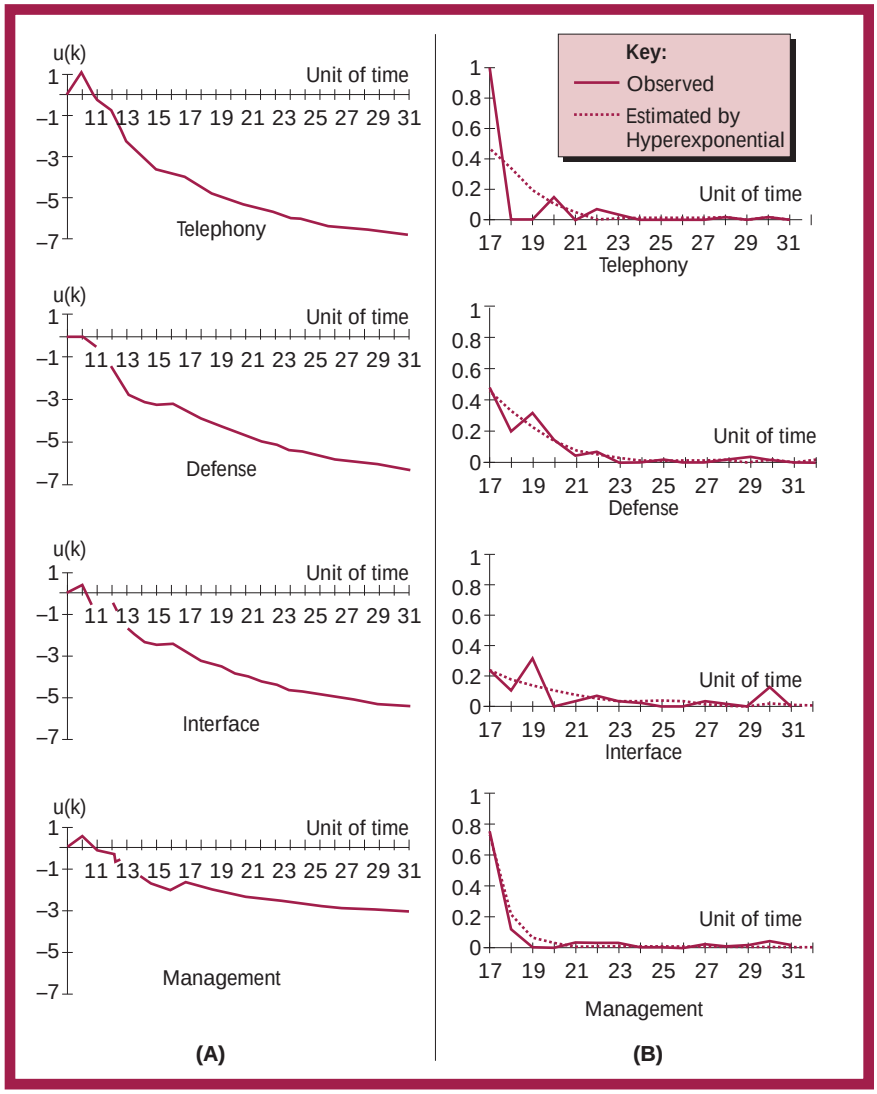

Figure 5. *Laplace factors (A) and failure intensities (B) for the four main components of the electronic switching system we studied.*

TABLE 2
RESIDUAL FAILURE RATES AND AVERAGE FAILURE RATE OVER THE LAST 10 MONTHS

	Estimated residual failure rate	Observed average (last 10 units of time)
Telephony	1.2×10^{-6}/h	1.0×10^{-5}/h
Defense	1.4×10^{-5}/h	1.6×10^{-5}/h
Interface	2.9×10^{-5}/h	3.7×10^{-5}/h
Management	8.5×10^{-6}/h	2.0×10^{-5}/h

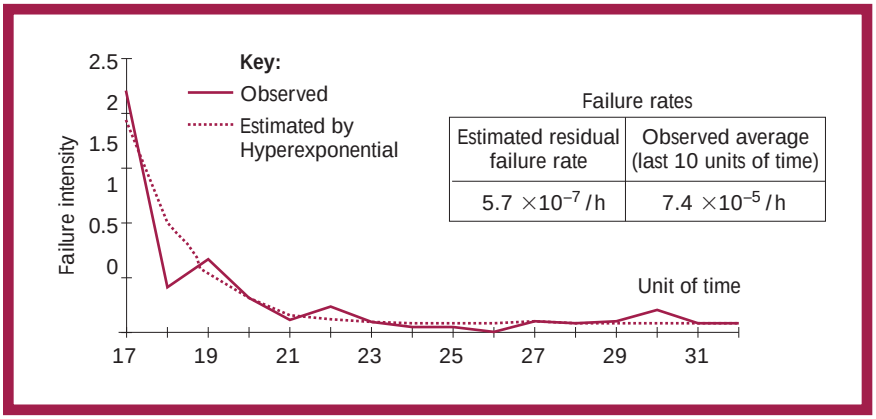

Figure 6. *Failure rate and failure intensity observed and estimated by the hyperexponential (HE) model for the whole software.*

Trend analysis. Figure 3A plots the Laplace factor for the whole data set when considering all the installed ESSs. Between time units 4 and 6, declining reliability occurred as a result of changes in the nature of the validation phase tests because new parts of the programs were activated. The software was put into operation before observing a noticeable reliability growth. Reliability growth took place only during the system's operational life.

Evaluating the Laplace factor for FRs recorded during system operation leads to Figure 3B, which shows reliability fluctuation from time units 14 to 24, which was not obvious when considering the whole data set. This fluctuation confirms that removal of a part of the data corresponding to declining reliability leads to negative Laplace factor values and amplifies the variation in the trend. This fluctuation is due mainly to the installation of 40 ESSs during this period.

Analyzing an average system by dividing the failure intensity by the number of installations in service leads to Figure 4: the curve obtained is smooth compared to that of Figure 3. Reliability fluctuation between time units 14 and 17 corresponds to the introduction of six ESSs. Reliability growth tends to be regular from time unit 17 onwards. This behavior is also apparent in the four components, as shown by the Laplace factor displayed in Figure 5A: all the components exercised reliability fluctuation between time units 14 and 16, with the management component most sensitive to change.

Reliability growth models. We first adopt the customer's perspective: evaluation of the software's reliability in operation. Then, we consider the supplier's perspective: to plan the maintenance effort, we estimate the number of failures expected to occur during the subsequent time period.

Software and component reliability evaluation. When the software is in operation, one of the main measures of interest is the residual failure rate corresponding to the steady behavior of the software. The hyperexponential (HE) model[11] is the only one that lets us estimate this measure, so we will use this model to evaluate the residual failure rate and the failure intensity.

The results of the trend analyses in Figures 4 and 5A show regularity from time unit 17 on. These results guide the model application: the failure intensity and residual failure rate are evaluated using failure data from time unit 17. Figure 5B shows the results of applying HE to each component separately, which gives the observed and estimated failure intensities. Table 2 gives the residual failure rates obtained from applying HE.

To check the validity of the results, we evaluated the average failure rate observed during the last 10 time units of operation, as shown in the last column of Table 2. As expected, this average is higher than the residual for all components. However, it is very close to the latter for the defense and interface components. This means that these components had almost reached a steady behavior during the last 10 time units, whereas the telephony and management components were still evolving. Figure 6 gives the results obtained from applying HE to the whole software.

If we review figures 5B and 6 and

Table 2, we can conclude the following:

♦ The evolution of the whole-failure intensity masks the real evolution of the component intensities.

♦ The failure rate obtained by summing over the component residual failure rates (5.3×10^{-5}/h) is close to the residual failure rate of the software (5.7×10^{-5}/h), which means that even though the components are not totally independent, the residual failure rate of the software is almost equal to the sum of the failure rates of its components.

♦ The residual failure rate may be regarded as high when compared to those of other systems. However, this failure rate must be moderated by acknowledging that the severity of failures is not distinguished and the estimated value includes failures with minor consequences as well. Unfortunately, the FRs do not record failure severity.

Maintenance planning. We adopt here the supplier's and maintainer's perspective: the maintenance effort is a function of the average effort to make a correction and the number of corrections to be performed. To do this, we consider data collected from all the installed systems.

Figure 3B indicates global reliability growth over the operational life and suggests the application of models with decreasing failure intensity: we use HE and the exponential model[12] (EXP) for data pertaining to operation. However, we also present the results of an S-shaped model[13] (SS) to allow for comparison and highlight the advantage of analyzing trends before applying models. We use failure data observed during time units 9 to 19 to predict the number of failures that will occur during the rest of the observation period. Figure 7 shows the results. As expected, HE and EXP give good results, whereas SS is overly optimistic. Compared to the 34 observed failures for this period, HE predicted 37, EXP predicted 33, and SS predicted only 9.

Nevertheless, Figure 3A suggests applying the SS model from the beginning, to include the trend change around time units 5 to 9. This is more in line with the model's assumptions. Doing so gives the results shown in Figure 8, including a significant improvement in the model's prediction accuracy.

The results show how much the use of trend test results can improve predictions when we apply the models to data exhibiting behavior according to the models' assumptions. Moreover, we can obtain equivalent and good results for maintenance planning over the next year if the models we use are applied to subsets of data displaying a trend in accordance with the models' assumptions.

Software reliability now faces a paradox: although software is the current bottleneck to achieving dependable computer systems, current

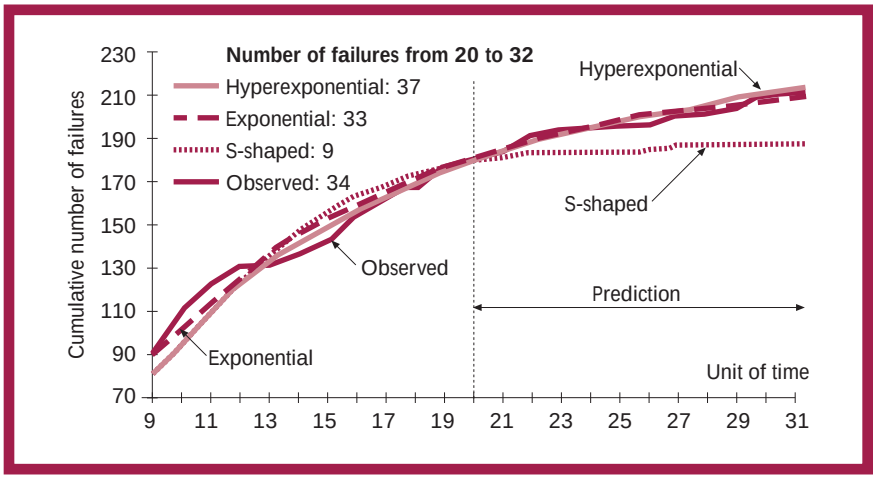

Figure 7. *Cumulative number of failures observed and estimated by the models using the operational data set.*

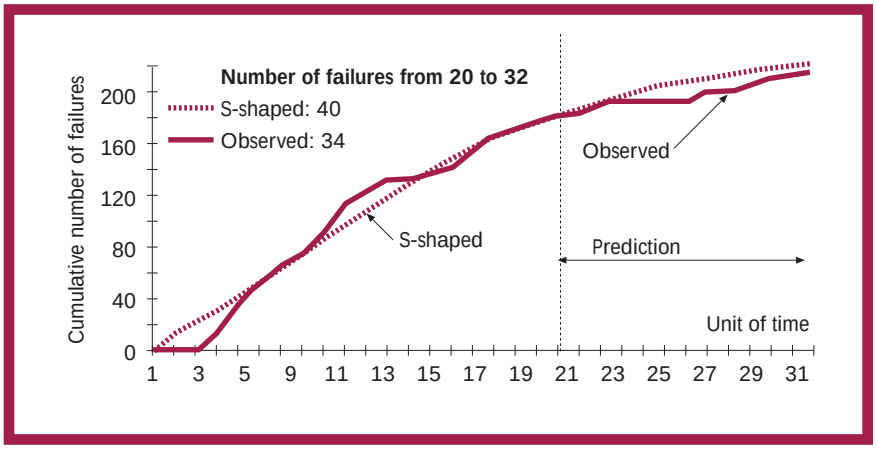

Figure 8. *Cumulative number of failures estimated by the SS model using the whole data set.*

reliability practice does not generally take software reliability into account. In part this is because reliability evaluation has often been restricted to the application of reliability models to failure data and in many cases those applications failed to give accurate predictions.

Our method has successfully helped analyze the software reliability of several real systems. We believe it is mature enough to be integrated into a more general approach to software reliability engineering and to be used in real time to manage software development. The approach we propose provides some solutions to the current software reliability evaluation paradox. However, some limits in the current state of the art must still be overcome to allow efficient application of software evaluation in the development process. We must tie software reliability more strongly to the earlier part of the development process and correlate the observed reliability with some characteristics of this process to identify significant areas for reliability improvement. ◆

ACKNOWLEDGMENT

We thank the company that provided the data analyzed in this article and, especially, the engineers who participated in that analysis. We also thank Sylvain Metge for his contribution to the development of Sorel. Our work was partially supported by the European ESPRIT Long Term Research Project 20072: Deva (Design for Validation).

REFERENCES

1. M. Kaâniche, K. Kanoun, and S. Metge, "Failure Analysis and Validation Monitoring of a Telecommunication Equipment Software System," *Annales des Telecommunications*, Vol. 45, Nos. 11-12, 1990, pp. 657-70.
2. V.R. Basili and D.M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Trans. Software Eng.*, Vol. 10, No. 6, 1984, pp. 728-38.
3. Y. Levendel, "Reliability Analysis of Large Software Systems: Defects Data Modeling," *IEEE Trans. Software Eng.*, Vol. 16, No. 2, 1990, pp. 141-52.
4. R. Chillarege and S. Biyani, "Identifying Risk Using ODC Based Growth Models," *5th Int'l Symp. Software Reliability Eng.*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 282-288.
5. R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, Englewood Cliffs, N.J., 1987.
6. O. Gaudoin, "Optimal Properties of the Laplace Test for Software-Reliability Models," *IEEE Trans. Reliability*, Vol. 41, No. 4, 1992, pp. 525-32.
7. K. Kanoun and J.-C. Laprie, "Software Reliability Trend Analyses: From Theoretical To Practical Considerations," *IEEE Trans. Software Eng.*, Vol. 20, No. 9, 1994, pp. 740-747.
8. N. Adams, "Optimizing Preventive Service of Software Products," *IBM J. Research and Development*, Vol. 28, No. 1, 1984, pp. 2-14.
9. K. Kanoun and T. Sabourin, "Software Dependability of a Telephone Switching System," *17th IEEE Int'l Symp. Fault-Tolerant Computing*, IEEE, Pittsburgh, 1987, pp. 236-41.
10. K. Kanoun, M. Kaâniche, and J.-C. Laprie, "Experience in Software Reliability: From Data Collection to Quantitative Evaluation," *4th Int'l Symp. Software Reliability Eng.*, IEEE CS Press, Los Alamitos, Calif., 1993, pp. 234-245.
11. J.-C. Laprie et al., "The KAT (Knowledge-Action-Transformation) Approach to the Modeling and Evaluation of Reliability and Availability Growth," *IEEE Trans. Soft. Eng.*, Vol. 17, No. 4, 1991, pp. 370-82.
12. A. L. Goel and K. Okumoto, "Time-Dependent Error Detection Rate Model for Software and Other Performance Measures," *IEEE Trans. on Reliability*, R-28, 1979, pp. 206-11.
13. S. Yamada, "Software Quality/Reliability Measurement and Assessment: Software Reliability Growth Models and Data Analysis," *J. Information Processing*, Vol. 14, No. 3, 1991, pp. 254-66.

Karama Kanoun is chargee de recherche of CNRS, the French National Organization for Scientific Research. 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, including hardware and software. Kanoun has completed several research contracts and has been consultant for several French companies and for the International Union of Telecommunications. She also served as a program committee cochair of the International Symposium on Software Reliability Engineering, as vice-chair of the Second International Conference on Reliability, Maintainability, and Safety, and as general chair of ISSRE '95. She is a member of the working group of the European Workshop on Industrial Computer Systems, Technical Committee 7: Reliability, Safety, and Security, and a member of the Dependability of Computing Systems AFCET working group.

Kanoun received a CE from the French National School of Civil Aviation and PhDs in engineering and science from the National Institute Polytechnique of Toulouse. Her doctorate of science thesis dealt with the theoretical and practical aspects of software dependability growth characterization, modeling, and evaluation. She is a member of the IEEE Computer Society, ACM, and AFCET.

Mohamed Kaâniche is charge de recherche at CNRS. He joined the LAAS-CNRS's Fault Tolerance and Dependable Computing group in 1988. His current research interests include dependability modeling and computing systems evaluation with a focus on software reliability growth evaluation and operational systems' security assessment. He also works on the definition of process models for the development of dependable systems.

Kaâniche received a CE from the French National School of Civil Aviation and a PhD in computer science and automation from the University of Toulouse. He is a member of AFCET.

Jean-Claude Laprie is directeur de recherche of CNRS, the French National Organization for Scientific Research. He joined LAAS-CNRS in 1968 and directed the research group on Fault Tolerance and Dependable Computing from 1975 to 1996. His research has focused on dependable computing since 1973, and especially on fault tolerance and on dependability evaluations, subjects on which he has authored and coauthored more than 100 papers, as well as coauthored or edited several books. He also founded and directed, from 1992 to 1996, the Laboratory for Dependability Engineering, a joint academia-industry laboratory. In 1984 and 1985, Laprie was chairman of the IEEE Computer Society Technical Committee on Fault-Tolerant Computing, and he was chairman of IFIP WG 10.4 on Dependable Computing and Fault Tolerance from 1986 to 1995.

Laprie received a CE from the Higher National School for Aeronautical Constructions, Toulouse, a PhD in engineering in automatic control, and a PhD in computer science from the University of Toulouse. He is a member of the IEEE Computer Society, ACM, and AFCET.

Address questions about this article to Kanoun at Centre National De La Recherche Scientifique, Laboratoire d'Analyse et d'Architecture des Systemes, 7, avenue du Colonel Roche, 31077 Toulouse Cedex 4, France; kanoun@laas.fr.