Field failures and reliability in operation

Karama Kanoun, Ram Chillarege, Ravishankar Iyer, Jean-Claude Laprie, John Musa

Panel session on:

Field Failures and Reliability in Operation

Moderator : Karama Kanoun (LAAS-CNRS, Toulouse, France)

Panelists : Ram Chillarege (IBM T. J. Warson Research Centre) **, Ravishanker K. Iyer** Univ. of Illinois at Urbana Champaign)**, Jean-Claude laprie** (LAAS-CNRS, Toulouse)**, John Musa** (AT&T Bell Lab, Murray Hill, NJ)

Introduction
Karama Kanoun

The majority of the reported work related to software reliability focuses mainly on development and validation. Reliability in operation is however the principal concern of the users (or customers) and is thus of prime importance. Field measurements provide information that allows the effect of errors on software behavior to be understood. It provides accurate information a) on the system being observed, b) for the elaboration and validation of analytical models and c) for the improvement of the development process.

The data collected helps to explain and characterize the software under study. Qualitative analysis of the failure, error and fault types observed in the field yields feedback to the development process and can thus contribute to improving the production process.

In general, error analysis:

- allows identification of the various causes of outages as well as the main factors that influence the software behavior;
- draws attention to potential problem areas with a new application;
- gives insight into the different ways of handling software development;
- provides useful information which is crucial to guide research and development.

Multiple installations provide millions of hours of operation and, as such, supply genuine statistical failure data allowing:

- statistically sound estimates of reliability measures, such as the failure rate or the mean time to failure;
- sufficient experimental data, providing the means for an accelerated feedback to the development process.

The development of realistic models to describe the failure behavior of the software is essential and the validation of these models needs to be based on real data from actual measurements. Measurements allow identification of the most influent parameters (such as the workload or the operational profile) in order to incorporate them in the models.

Analysis of failure data has to be carried out as soon as the data are available otherwise the results may be obsolete for the system *under study*. Early analysis also allows the project team to validate their own data. However, it is worth noting that analysis of failure data collected on a given system is beneficial for the development of the *following generations* of systems within the same application area.

Finally, during the panel discussion, both the supplier and the customer viewpoints will be addressed, taking alternatively the academic and the industrial perspective. Emphasis will be put on experience in operational reliability measurement and the kind of development improvement resulting from measurement feedback.

Reliability growth with ODC
Ram Chillarege

Traditional views on software reliability growth treat software defects as homogeneous with most of the discussion revolving around the abscissa of the model. Discussions worry whether the abscissa should be execution time, calendar time or percent of tests covered with little discussion on the focus of what is being modeled, namely the defect. A key study, [Chillarege 91], uncovered the impact of the defect types on the overall reliability growth experienced. This study showed that there is a strong relationship between the overall reliability growth experienced and the type of defects contributing to the growth. This lead to work on Orthogonal Defect

Classification (ODC) [Chillarege 92] which exploits defects to gain insight into the process and explain the progress of a product through the process. In ODC defects are classified into a small number of categories that extract the semantics of a defect. For the purposes of this discussion we need to focus on one of those attributes called the *defect type*. Defect types are chosen to identify the meaning of the fix and apply almost consistently throughout the life cycle of the development process. The idea in the classification is that the distribution formed by these categories changes as the product moves through the development process. Thus, the defect type distribution measures the progress of a product through the process. There are, however, necessary and sufficient conditions for the choice of these categories which makes such measurement possible.

One of the applications of ODC data is to use the defect type classification to model reliability growth for specific types. Since these categories are carefully chosen, and represent an independent and orthogonal activity in the process, the world of software reliability growth modeling can exploit the strength of being able to study the relative growth corresponding to activities in development. For instance, functional defects are commonly associated with design and should mostly be found early in the process. Thus the growth model for functional defects, should mature hopefully sooner than assignment and checking type defects, that are mostly code related. Developing reliability growth models by the semantics of defect type category give us further insight into what parts of development are more mature than others, aiding in quantitative risk assessment.

By contrast, to the traditional reliability modeling, ODC throws a new light on the problem and techniques to date. It also puts the focus back on the software defect which, in the first place, should be the point of attention. Clearly, there are several issues that need to be better understood and researched to refine a classical technique such as growth modeling. However, I believe this is one of the first steps that has the potential to build, more general cross product and cross process models which could be reasonably calibrated. ODC has been deployed in the IBM Corporation quite extensively today. Our efforts in deploying ODC began more than two and one half years ago, and today have ten different labs use it in their projects with some of them stepping it up to production use. In this panel, I will raise some of the issues that are open for research.

References

[Chillarege 91] R. Chillarege, W.L. Kao, and R. Conduit, "Defect Type and its Impact on the Growth Curve", Proc. *13th International Conference on Software Engineering*, May 1991.

[Chillarege 92] R. Chillarege et. al., "Orthogonal Defect Classification for Defect Control", *IEEE Transactions on Software Engineering*, November 1992.

Measurement and Analysis of Software Failures
Ravishanker K. Iyer

When a computer system is in normal operation, various errors occur in both the hardware and the software. There are many possible sources of these errors, including untested manufacturing faults and software defects, wearing out of devices, transient errors induced by radiation, power surges, or other physical processes, operator errors, and environmental factors. The occurrence of errors is also highly dependent on workloads running on the system.

Given field error data collected from a real system, a measurement-based study typically, consists of four steps: 1) data processing, 2) model identification and, 3) model solution if necessary, and 4) analysis of models and measures. With the rapid improvement in hardware reliability, the contribution of software to overall system dependability has become significant. Recently, [Gray 90] showed that software accounts for 60% of unplanned outages in Tandem systems. Two independent studies [Iyer 85, Sullivan 91] showed a similar trend in IBM systems. With the projected increase in the size and the complexity of software, this trend is likely to continue.

During the past fifteen years there has been considerable emphasis on the use of field measurements to quantify the dependability of large continuously evolving software systems. [Endres 75] analyzed software error data collected from the DOS/VS operating system during the testing phase. In [Thayer 78] a wide-ranging analysis of software error data collected during the development phase, is described. [Basili 84] analyzes the relationships between the frequency and distribution of errors during software development, the maintenance procedures, and a variety of environmental factors. In [Chillarege 92], an "orthogonal defect classification" scheme is proposed. The scheme has been used to provide effective feedback on the development process.

Using test and failure data from AT&T switching system software, [Clapp 92] related the software failure characteristics to the system functional-structure. The methodology was intended to support the reduction of testing cost and enhancement of software quality by improving test selection, eliminating test redundancy, and identifying error-prone source files. The results showed that once the black-box tests had been designed and developed, white-box methods can be used to map the test-data to the product internals and hence guide future testing efforts.

Extensive analyses of operational software have also been performed in the past decade. Two early studies [Castillo 82,

Rossetti 82] proposed a workload-dependent probabilistic model to predict software errors based on measurements from DEC and IBM systems. [Velardi 84, Iyer 85] evaluated the effectiveness of error recovery routines and addressed the issue of hardware-related software errors using failures and recovery data from the MVS operating system. The measurements showed that the system fault tolerance almost doubled when recovery routines are provided. The results also showed that 25% to 35% of all software failures were hardware-related. The system failure probability for hardware-related software errors was measured to be three times that due to software errors in general. In [Hsueh 87], a semi-Markov model was constructed from data, to describe software error occurrence and recovery in the MVS environment. [Sullivan 91] investigated software defects and their impact on system availability. The study focused on the categorization of defect types, defect-triggering events, and failure symptoms.

Recently, [Lee 93a] investigated the effect of faults in system software on overall system dependability, using failure reports from a Tandem GUARDIAN operating system. The study addressed several issues including: 1) software fault tolerance of process pairs, 2) recurrences, 3) error propagation, and 4) the resulting effects on software reliability.

The results showed that 72% of reported field software failures were recurrences of known software faults. The measured system tolerated nearly 80% of the reported software faults, thus demonstrating the effectiveness of hardware fault tolerance techniques against many software faults. [Lee 93b] developed a methodology to quantify relative and absolute improvement in the system fault tolerance due to the implemented software techniques. The approach was demonstrated via comparison of three major operating systems: the Tandem GUARDIAN, the VAX/VMS, and the IBM MVS system.

A number of methods, developed to analyze field data, have been incorporated into MEASURE+, an automated environment that allows, wide ranging analysis field failure data [Tang 93]. Given measured data from real systems in a specified format, MEASURE+ can generate appropriate dependability models and measures which accurately reflect system behavior in real environments. The failure behavior of the system is mapped into a small number of states with associated characteristics. This mapping is valuable for understanding actual error/failure characteristics, identifying dependability bottlenecks, evaluating dependability of real systems, and verifying assumptions made in analytical models.

It is clear that the need for reliable software continues to pose formidable challenges. What is lacking is the availability of accurate evaluation methods and tools that can determine, how well current methods work and provide effective feedback to designers. Further, it is critically important that interactions between software and hardware be taken into account and the dependability of the software be judged by its contribution to overall system dependability.

References

[Basili 84] R V R. Basili and B. T. Perricone, "Software errors and complexity: An empirical investigation", Communications of the ACM, Vol. 22, No. 1, pp. 42-52, Jan. 1984.

[Castillo 82] X. Castillo and D. Siewiorek, "A Workload Dependent Software Reliability Prediction Model," 12th *Int. Symp. on Fault-Tolerant Computing*, Santa Monica, Ca., June, 1982.

[Chillarege 92] R. Chillarege et al., "Orthogonal Defect Classification Concept for In-Process Measurements," *IEEE Trans. Software Engineering*, Vol. 18, No. 11, Nov. 1992, pp. 943-956.

[Clapp 92] R K. Clapp and R. K. Iyer, "Analysis of Large System Black-Box Test Data," Proc. *Third Int. Symp. Software Reliability Engineering*, pp. 94-103, Oct. 1992.

[Endres 75] R A. Endres, "An Analysis of Errors and Their Causes in System Programs," Proc. *Int. Conf. Software Engineering*, pp. 327-336, Apr. 1975.

[Gray 90] J. Gray, "A Census of Tandem System Availability between 1985 and 1990," *IEEE Trans. Reliability*, Vol. 39, No. 4, Oct. 1990, pp. 409-418.

[Hsueh 87] M.C. Hsueh and R.K. Iyer, "A Measurement-Based Model of Software Reliability in a Production Environment," Proc. *11th Annual Int. Computer Software & Applications Conf.*, pp. 354-360, Oct. 1987.

[Iyer 85] R.K. Iyer and P. Velardi, "Hardware-Related Software Errors: Measurement and Analysis," *IEEE Trans. Software Engineering,* Vol. SE-11, No. 2, pp. 223-231, Feb. 1985.

[Lee 93a] I. Lee and R.K. Iyer, "Measurement-Based Reliability Analysis of the Tandem GUARDIAN Operating System," submitted for review to *IEEE Trans. Software Engineering*.

[Lee 93b] I. Lee, D. Tang, R. K. Iyer, and M.-C. Hsueh, "Measurement-Based Evaluation of Operating System Fault Tolerance," to appear in *IEEE Trans. Reliability*, June 1993.

[Rossetti 82] D.J. Rossetti and R.K. Iyer, "Software failures on the IBM-3081", *COMPSAC-82*, Chicago, October 1982.

[Sullivan 91] M.S. Sullivan and R. Chillarege, "Software Defects and Their Impact on System Availability — A Study of Field Failures in Operating Systems," Proc.

21st Int. Symp. Fault-Tolerant Computing, pp. 2-9, June 1991.

[Tang 92] D. Tang and R.K. Iyer, "Analysis of the VAX/VMS Error Logs in Multicomputer Environments — A Case Study of Software Dependability," Proc. *Third Int. Symp. Software Reliability Engineering*, pp. 216-226, Oct. 1992.

[Tang 93] D. Tang and R.K. Iyer, "MEASURE+ — A Measurement-Based Dependability Analysis Package", *ACM SIGMETRICS*

Conference on Measurement and Modeling of Computer Systems, Santa Clara, California, May 1993.

[Thayer 78] T.A. Thayer, M. Lipow, and E.C. Nelson, *Software Reliability*, North-Holland Publishing Company, 1978.

[Velardi 84] P. Velardi and R.K. Iyer, "A Study of Software Failures and Recovery in the MVS Operating System", *IEEE Trans. Computers*, Vol. C-33, No. 6, pp. 564-568, June 1984.

On the temporary character of operation-persistent software faults

Jean-Claude Laprie

Temporary faults have long been recognized as constituting the vast majority of hardware faults, and the progresses in hardware integration can only emphasize this tendency (see e.g., the field data from several sources in [Siewiorek 92]). A direct consequence is the emphasis placed in the design of fault-tolerant systems on discriminating between temporary and permanent faults: the misinterpretation of a temporary fault as a permanent fault results in an unnecessary decrease in the available redundancies, thus in lowering dependability.

Temporary faults are not limited to hardware: the notion of temporary fault applies to software as well. Although such a notion has been introduced a long time ago [Elmendorf 72], and more recent studies have shown that most of the software faults present during operational life are temporary faults [Gray 86], the very notion of temporary software fault is often felt as contradicting our perception of software. In fact, if it is not arguable that the ultimate cause of software faults are present as long as they are not fixed, it has to be recognized that most software faults manifesting in operation in large, complex, software are subtle enough in order that their activation conditions depend on equally subtle combinations of internal state and external solicitation, so that they can hardly be reproduced. Stated in other terms, the failure domain in the input space of software faults which persist in operation can vary with the conditions of execution of the software, and be a null space under most operating conditions.

Acknowledging that operation-persistent software faults are temporary is likely to have dramatic consequences on the design for dependability of software systems, via their structuring into self-checking components [Yau 75, Blum 89, Laprie 90] which incorporate measures for error detection in addition to functional code. Several strategies for error treatment (following error detection) could then be implemented, from backward recovery via recovery points, to exception handling in order to prevent the failure of a task to lead to system failure. Although such strategies for software-fault tolerance have already been implemented in some systems [Gray 86, Huang 93], they are far from large adoption. There is however a real need for providing software with fault tolerance, as software is currently recognized as the current bottleneck in terms of dependability [Gray 90, Cramp 92]. Recognizing the temporary character of software faults would enable software fault tolerance not to be restricted to design diversity, and thus to widen its field of application, which is currently mostly limited to safety-critical applications because of the high cost of design diversity.

References

[Blum 89] M. Blum, S. Kannan, "Designing programs that check their work", Proc. *ACM Symposium on Theory of Computing*, 1989, pp. 86-97.

[Cramp 92] R. Cramp, M.A. Vouk, W. Jones, "On operational availability of a large software-based telecommunications system", Proc. *3rd Int. Symp. on Software Reliability Engineering*, Research Triangle Park, North Carolina, Oct. 1992, pp. 358-366.

[Elmendorf 72] W.R. Elmendorf, "Fault-tolerant programming", Proc. *2nd IEEE Int. Symp. on Fault Tolerant Computing (FTCS-2),* Newton, Massachusetts, June 1972, pp. 79-83.

[Gray 86] J.. Gray, "Why do computers stop and what can be done about it?", Proc. *5th Symp. on Reliability in Distributed Software and Database Systems,* Los Angeles, Jan. 1986, pp. 3-12.

[Gray 90] J. Gray, "A census of Tandem system availability between 1985 and 1990", *IEEE Trans. on Reliability*, vol. 39, no. 4, Oct. 1990, pp. 409-418.

[Huang 93] Y. Huang, C. Kintala, "Software implemented fault tolerance: technologies and experience", Proc. *23rd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-23),* Toulouse, June 1993, pp. 2-9.

[Laprie 90] J.C. Laprie, J. Arlat, C. Beounes, K. Kanoun, "Definition and analysis of hardware- and software-fault-tolerant architectures", *IEEE Computer,* vol. 23, no. 7, July 1990, pp. 39- 51.

[Siewiorek 92] D.P. Siewiorek, R.S. Swarz, *The Theory and Practice of Reliable System Design,* Digital Press, 1992.

[Yau 75] S.S. Yau, R.C. Cheung, "Design of self-checking

software", Proc. *Int. Conf. on Reliable Software,* Los Angeles, CA, April 1975, pp. 450-457.

Position statement
John D. Musa

The principal task in the field is to monitor field reliabilities against *objectives*. We deliberately emphasize the plural because we may be tracking reliabilities of key components as well as the reliability of the total system, AND we usually track reliabilities for different failure severity classes.

Monitoring requires that we collect failure *and* operational profile data. It is best, of course, that we collect complete failure data from all installations. This is probably economically feasible only if we instrument the delivered software so that failures are detected automatically. This has the disadvantage that types of failures that are not foreseen cannot be detected. However, if the ratio of automatically-to-manually-detected failures remains

reasonably stable, we can estimate total failures from automatically-detected ones. If we must record failures manually, we will probably have to select a random sample of installations. It is important that we collect actual *failure*, not *fault,* data; manual trouble reports often represent faults. Each recurrence of a failure must be reported if we are to accurately access customer impact and level of customer satisfaction.

We need to collect operational profile data to determine if we accurately estimated how the system was going to be used. An inaccurate operational profile is a possible cause of variation in software reliability achieved.

When we compare reliabilities achieved with objectives, we note discrepancies and try to find their causes. In addition to an inaccurate operational profile, other possibilities include differences in failure definition, errors in data collection, and variations from the planned development process.

The primary purpose of the monitoring is to identify both product and process improvements to help us better or more efficiently meet the reliability requirements. Product improvements are typically introduced in near term releases. Process improvements may be introduced over a wider time span and usually a wide range of projects.

We not only check actual reliabilities achieved but we also survey the level of satisfaction of customers with the products. This can highlight situations where reliability objectives are not being correctly set.