Software Quality and Reliability Basics

John D. Musa
AT&T Bell Laboratories
Whippany, NJ 07981

ABSTRACT

The importance of software quality and the relationship of software reliability to software quality are discussed. The need for software reliability measures is demonstrated by outlining some possible applications. Basic software reliability concepts are presented, including software modeling.

Why Measure?

Increasing global competition and high development costs have intensified the pressures to quantify software quality and to measure and control the level of quality delivered. Software reliability is the most important and most measurable aspect of software quality and it is very customer oriented. It is a measure of how well the program functions to meet its operational requirements.

Why is this quantification of quality so important? Almost all of the institutions (airlines, banks, manufacturers, universities, etc.) that use software in their operations find themselves facing sharply increasing international competition. As our global society becomes more dependent on information (as contrasted to capital or labor) in the production of goods and services, the pressures for higher quality, lower cost, and faster delivery for software products are increasing. And we have more software developers eager to compete for customers. But software quality, cost, and schedules are conflicting characteristics; you can't have one without paying with another. Schedules and cost have tended to dominate until now because they can be concretely measured and specified. We have lacked a clear measure of quality.

The foregoing pressures have made a quantitative measure of quality necessary for both software developer and customer. Such a measure, plus the understanding of how it interacts with costs and schedules (i.e., a model) makes precise tradeoffs between goals possible. It enables you to plan more accurately for the resources you will need and to lay out schedules with greater confidence. Finally, better planning and better measurement lead to better visibility of the software development progress. Consequently, you can monitor development progress more accurately and exercise better control over it.

How Can Software Reliability Measures Help You?

Users of software reliability measures have found [1] that developer-customer dialog is substantially enhanced. It is necessary to define "failure" for the system concerned. This definition is, in effect, a negative specification of requirements, and it generally leads to a clarification for everyone of what these requirements are. Reliability figures can readily be related to the operational costs of failure. Thus the customer comes to understand the *real* reliability requirements of the system in question. Similarly, the developer can relate reliability level requested to development costs. Thus, the stage is set for negotiation of an optimum solution for the customer of the sum of capital (purchase price of the system) and operational costs. By increasing the precision with which the customer's needs are met, productivity in the broadest sense is enhanced.

Software reliability measures guide the developer to better decisions. In the system engineering stage, they promote quantitative specification of design goals, schedules, and resources required. They let you determine quality level during test and thus provide the means for evaluating the effect of various actions on quality so that it can be controlled. The measures also help in the better management of project resources.

The user will also benefit from software reliability measures, because the user is concerned with efficient *operation* of the system. If operational needs with respect to quality are inaccurately specified, the user will either get a system at an excessively high price or with an excessively high operational cost.

The models associated with software reliability measurement structure and enhance both developer and customer understanding of software quality and the factors affecting it. The factors include the time the program has been executing, software *product* characteristics, development *process* characteristics (including resources), and the *operational environment* or ways in which the software is used. These models permit the prediction, during test, of when various levels of quality will be obtained. Thus, once a quality objective has been chosen, release date can be predicted.

Developer and user, through accurate specification of what is a failure and what failure rate (or quality level) is optimum, can each increase customer satisfaction, provided the specification is met. The improved reputation resulting from high levels of customer satisfaction generally leads to a greater market share and higher profitability.

Basic Concepts

Software reliability is defined as the probability of failure-free operation of a computer program for a specified time in a specified environment. For example, a program might have a reliability of 0.82 for 8 hours of execution. A *failure* is a departure of program operation from requirements. *Failure intensity*, an alternate way of expressing software reliability, is defined as failures occurring with respect to some time unit. An expression equivalent to the reliability figure given above is that a program has a failure intensity of 0.025 failures per hour of execution. A *fault* is a defect in a program that causes a failure.

Software reliability is influenced by *fault introduction* resulting from new or modified code, *fault removal* that occurs in debugging, and the *environment* or ways in which the program is used. As a program is executed, failures will occur. If fault removal actions are taken (however imperfectly) in response to the failures, failure intensity will decrease as a function of time. Software reliability models characterize this change, as shown in Figure 1. A number of models have been developed [2-10]; see [1] for a classification and comparison of the models.

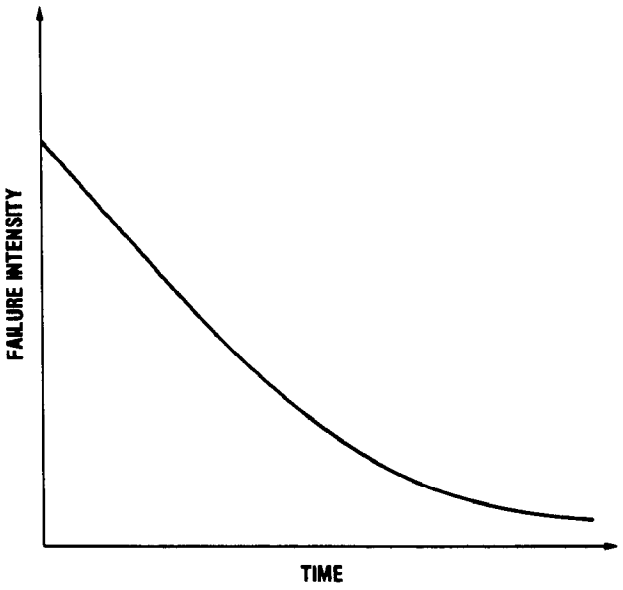

Figure 1. Software reliability model

Applications and State of the Art

Many applications for software reliability measurement have been developed, and considerable experience has been gained in its use [1]. We are now at the point where practicing software engineers in industry are independently testing the technology. This panel session presents a small sample of this work. It is not intended to be a comprehensive survey of applications. Rather than try to present such a survey or evaluate the state of the art, we will let the practitioners speak for themselves.

References

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

[2] Z. Jelinski and P.B. Moranda, "Software reliability research," in *Statistical Computer Performance Evaluation,* W. Freiberger, Ed. New York: Academic Press, 1972, pp. 465-484.

[3] P. Moranda, "Predictions of software reliability during debugging," in *Proc. Ann. Reliability and Maintainability Symp.* (Washington, DC), pp. 327-332, Jan. 1975.

[4] M. Shooman, "Probabilistic models for software reliability prediction," in *Statistical Computer Performance Evaluation,* W. Freiberger, Ed. New York: Academic Press, 1972, pp. 485-502.

[5] G.J. Schick and R.W. Wolverton, "Assessment of software reliability," presented in 11th Annual Meeting of German Operations Research Society, Hamburg, Germany, Sept. 6-8, 1972; in *Proc. Operations Research,* Physica-Verlag, Wurzburg-Wien, 1973, pp. 395-422.

[6] J.D. Musa, "A theory of software reliability and its application," *IEEE Trans. Software Eng.,* SE-1(3), pp. 312-327, Sept. 1975.

[7] B. Littlewood and J.L. Verrall, "A Bayesian reliability growth model for computer software," *J. Roy. Stat. Soc. - Series C,* 22(3), pp. 332-346, 1973.

[8] B. Littlewood, "What makes a reliable program -few bugs, or a small failure rate?," *AFIPS Conf. Proc.,* 49, pp. 707-713, May 1980.

[9] A.L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Trans. Rel.,* R-28(3), pp. 206-211, Aug. 1979.

[10] J.D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," *Proc. 7th International Conference on Software Engineering,* Orlando, Florida, March 26-29, 1984, pp. 230-238.

Author

John D. Musa is Supervisor of Software Quality at AT&T Bell Laboratories, Whippany, N.J. He has participated in or managed a variety of software products. His technical background and interests include software reliability, software engineering, and human factors. He is principal author (with A. Iannino and K. Okumoto) of the pioneering book "Software Reliability: Measurement, Prediction, Application," McGraw-Hill, 1987. He is a Fellow of the IEEE, cited for "contributions to software engineering, particularly software reliability."