SOFTWARE RELIABILITY MODELING – WHERE ARE WE AND WHERE SHOULD WE BE GOING?

THE NEED FOR SOFTWARE RELIABILITY MODELING

It may be argued that software reliability metrics are needed, most importantly, because no field can really mature until it can be described in a quantitative fashion. However, there are also some very specific reasons for a quantitative approach to software reliability. One needs software liability figures in order to do a good job of system engineering: to examine the trade offs between reliability and cost and reliability and schedules, to determine what reliability figure optimizes overall life cycle costs, to plan allocation of resources, and to specify reliability to a contractor who is developing software for you. Another large area of application is project management, where software reliability measures are needed for progress monitoring, scheduling and investigation of managerial alternatives. The length of a test period and hence the overall length of a project is highly correlated with the reliability requirements for the project. Therefore, reliabilities are intimately tied up with schedules. Changes in resources available to the project affect both reliability and schedules and one can be exchanged for the other. Reliability metrics offer an excellent means of evaluating the performance of operational software and controlling changes to it. Since change usually involves a degradation of reliability, one may use reliability performance objectives as a means for determining when software changes can be allowed and perhaps even how large they can be. Finally, reliability is one of the important parameters that should be used in investigating the benefits (or lack of benefits) of proposed new software engineering technology.

SOFTWARE RELIABILITY FUNCTIONS

Hecht [1] has categorized software reliability functions into measurement, estimation and prediction. This classification is used in this paper with some modification and extension. Software reliability is defined as the probability that a program will execute without failure caused by software for a specified time in a specified environment. The term "failure" refers to an unacceptable departure from proper operation. The term "unacceptable" must be defined by the customer. The "measurement" of software reliability is based on failure interval data obtained by running the program in its actual operating environment. Software reliability "estimation" refers to the process of determining software reliability metrics based on operation in a test environment. It should be noted that estimation can be performed with respect to present or future reliability quantities. The term software reliability "prediction" refers to the process of computing software reliability quantities from program data which does not include failure intervals. Typically, software reliability prediction takes into account factors such as size and complexity of the program, and is normally performed during a program phase prior to test. Note that future estimation might be thought of by some as prediction; we are deliberately making a careful distinction in terminology.

The various applications of software reliability metrics are closely tied to the three functions that have just been defined. System engineering primarily relies upon prediction; project management, upon estimation; and operational software management and evaluation of software engineering technology, upon measurement.

J. Musa
Bell Lab.
1 of 11

SOFTWARE RELIABILITY MODELS

Most of the work that has been done in the field of software reliability falls in one of six categories: calendar time models, the execution time model, Bayesian models, semi-Markov models, deterministic models and input space approaches. The initial approach to software reliability was through calendar time models; that is, attempts were made to look of reliability phenomena such as failures, reliability, mean-time-to-failure (MTTF), etc. as functions of calendar time. These early models focused attention on the problem of software reliability and contributed many valuable concepts toward the further development of the theory. [2-5]

However, the failure-inducing stress placed on software is related closely to execution time (CPU time) and not calendar time. The execution time model [6-11] recognizes this fact. It has been extensively tested on more than 20 software systems and the validity of the assumptions made in deriving the model has been carefully examined. [12]

Littlewood and Verrall [13] have proposed a Bayesian model that is perhaps the most mathematically elegant of the software reliability models, but it is, unfortunately, difficult to understand, and computations based on it are lengthy and costly. A model that focuses specifically on the problem of imperfect fault correction has been developed by Goel and Okumoto [14]; it is based on a view of fault correction as a semi-Markov process. The concept of imperfect fault correction is incorporated in the execution time model in a simpler fashion. Deterministic models have been proposed [15, 16] but they have not been validated.

It would appear that deterministic models oversimplify the failure detection and correction process and are not efficient in using the information available to them. Bayesian models perhaps represent the other extreme, in that both failure intervals and failure process parameters are viewed as being random. The execution time model takes the intermediate approach of considering failure intervals random but failure process parameters as varying with execution time in a deterministic fashion.

A final viewpoint, the input space approach, is based on enumerating all the possible sets of input or environmental conditions for a program and determining the proportion of these that result in successful operation. Although this approach is theoretically appealing, the large number of possible input sets for any useful program makes it impractical. The counts would have to be weighed by run times and frequencies of operation for the various input sets, in order to provide results that would be compatible with hardware reliability theory.

EXECUTION TIME MODEL

The execution time model permits the development of relationships that indicate number of failures experienced and present MTFF as functions of execution time (see Figures 1 and 2). It relates total failures and initial MTFF to the number of faults in the system. An initial estimate of the number of faults, prior to testing, can be determined from the size (and perhaps complexity) of the program. A debugging process model is provided which relates execution time and calendar time and thus allows execution time quantities to be converted into dates. The model can be used to make predictions of the remaining number of failures to be experienced, the execution time and the calendar time required to reach a MTTF objective. If this objective is set as the

criterion for terminating the project, completion dates can be predicted. As testing proceeds, two of the key parameters of the model can be statistically reestimated from failure intervals experienced. This permits the estimation of a number of derived quantities such as present MTTF and estimated completion date. The estimates made are maximum likelihood estimates; confidence intervals are also calculated.

Most of the assumptions that were made in deriving the execution time model have been validated [12] and experience has been gained with the model on a wide variety of software systems (more than 20 as of this date). A program is available [17, 18] to handle the statistical calculations. Sample output from the program is shown in Figure 3.

User comments indicate that the execution time model provides a good conceptual framework for viewing the software failure process. It is simple, its parameters are closely related to the physical world and it is compatible with hardware reliability theory. Most users feel that the benefits currently exceed the costs, which are basically data collection and computation. There have been two interesting side benefits. The process of defining just what constitutes a failure and the process of setting a MTTF objective have both been salutary in opening up communication between customer and developer.

STATE OF THE ART AND RESEARCH NEEDS

Software measurement can presently be achieved with excellent accuracy. Figure 4 illustrates a software system in the operational state. The maximum likelihood estimate and 75% confidence bounds are indicated for present MTTF. Variations in MTTF and the size of the confidence interval are generally highly correlated with periods of fault correction or the addition of new capabilities.

The quality of software reliability estimation is dependent upon the representativeness of testing; hence good test planning is esesntial. If one desires to know the absolute value of the MTTF, knowledge of the test compression factor is necessary. The test compression factor relates the amount of time spent in test with the equivalent amount of operating time represented. It is known theoretically how to compute this number but the only practical approach at present is to estimate it from a similar porject in a similar test environment. Research activity in this area would definitely be beneficial. One might characterize the present quality of software reliability estimation as good for present estimation and fair for future estimation. Future estimation also requires, in addition to the factors previously listed, a number of resource parameters. Data collection to determine the values of these parameters and the extent to which they vary between different projects or different classes of projects is urgently needed. Figure 5 illustrates the variation in present MTTF as the system test phase of a project proceeded (maximum likelihood estimate and 75% confidence bounds are indicated). Although the accuracy of the absolute estimates is dependent on the test compression factor, the relative values (i.e., denoting progress) are highly accurate.

The function of software reliability prediction needs the most work. However, it also offers great promise in terms of ultimate potential benefits [9]. All of the input quantities required for software estimation are needed for this function as well. In addition, one requires the number of faults inherent in the software, the fault exposure ratio, the fault reduction factor and the linear execution frequency. Figure 6 indicates the quantities and relationships involved in software

reliability perdiction. The number of faults inherent in the software N_0 must be determined from estimates of program size and data on fault densities. Data on fault densities is just beginning to accumulate but much more is needed, along with information on the variation of the fault density with program complexity and other factors. The fault reduction factor B indicates the ratio of net faults repaired to failures detected. It is a function of the test or operational environment and appears to be constant across similar environments. The initial MTTF, T_0, must be predicted from total failures M_0, from the linear execution frequency of the program f (throughput divided by object program size) and the fault exposure ratio K. The fault exposure ratio is expected to be dependent on the dynamic structure of the program and the degree to which faults are data dependent. Further investigation of the properties of this ratio and the factors upon which they depend is very important if we are to obtain good absolute software reliability predictions. Relative predictions can be made without this knowledge in many cases and they may be useful for many system engineering studies.

CONCLUSIONS

Software reliability has come a long way since its early beginnings in 1972. Many of the early problems have been solved and a reasonable amount of actual failure data has been collected. It may be seen from this paper that a number of problems remain to be solved and that new problems will probably suggest themselves as the field progresses. It is important, however, that we build upon the results that have already been achieved so as to maximize the efficiency of our efforts.

REFERENCES

1. H. Hecht, "Measurement, estimation, and prediction of software reliability," In Software Engineering Technology – Volume 2, Infotech International, Maidenhead, Berkshire, England, 1977, pp. 209–224; also in NASA Report CR145135, 1977 Jan.

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

3. M. Shooman, "Probabilistic models for software reliability prediction," in Statistical Computer Performance Evaluation, see [2], pp. 485–502; also in 1972 Int: Symp. Fault-Tolerant Computing, Newton, Mass., 1972, June 21, pp. 211-215.

4. N. F. Schneidewind, "An approach to software reliability prediction and quality control," in 1972 Fall Joint Comput. Conf., AFIPS Conf. Proc., Vol. 41, Montvale, NJ: AFIPS Press, pp. 837–847.

5. G. J. Schick and R. W. Wolverton, "Assessment of software reliability," presented at 11th Annual Meeting of German Operations Research Society, Hamburg, Germany, 1972 Sep 6–8.

6. J. D. Musa, "A software reliability model," presented at NASA Software Engineering Workshop, Goddard Space Flight Center, Greenbelt, Maryland, 1977 Sep. 19.

7. J. D. Musa, "A theory of software reliability and its application," IEEE Trans. Software Engineering, Vol. SE-1, 1975 Sep., pp. 312-327.

8. J. D. Musa, "Software reliability measurement," in Software Phenomenology: Working Papers of the Software Life Cycle Management Workshop, Airlie, Va., 1977, Aug. 21-23, pp. 427-451. Also to be published in Journal of Systems and Software.

9. J. D. Musa, "Software reliability measures applied to system engineering," in 1979 NCC Proceedings, New York, N.Y., 1979 June 4-7, pp. 941-946.

10. J. D. Musa, "The use of software reliability measures in project management," in Proc. COMPSAC 78, Chicago, Ill., 1978 Nov. 14-16, pp. 493-498.

11. Patricia A. Hamilton and John D. Musa, "Measuring reliability of computation center software," in Proc. 3rd. Int. Conf. Soft. Eng., Atlanta, Ga., 1978 May 10-12, pp. 29-36.

12. J. D. Musa, "Validity of the execution time theory of software reliability," in IEEE Transactions on Reliability, Vol. R-28, No. 3, 1979 Aug., pp. 181-191.

13. B. Littlewood and J. L. Verrall, "A Bayesian reliability growth model for computer software," in 1973 IEEE Symp. Computer Software Reliability, New York, NY, 1973, Apr. 30-May 2, pp. 70-77.

14. A. L. Goel and K. Okumoto, Bayesian Software Prediction Models – An Imperfect Debugging Model for Reliability and Other Quantitative Measures of Software Systems, Rome Air Development Center Report RADC-TR-78-155, Vol. I.

15. H. Remus and S. Zilles, "Prediction and management of program quality," in Proc. 4th Int. Conf. on Software Engineering, Munich, Germany, 1979 Sep. 17-19, pp. 341-350.

16. I. Nathan, "A deterministic model to predict 'error free' status of complex software in development," in Proc. Workshop on Quantitative Software Models, Kiamesha Lake, N.Y., 1979 Oct. 9-11, to be published.

17. J. D. Musa, Program for software reliability and system test schedule estimation – user's guide, IEEE Computer Society Repository, Ref. No. R77-244.

18. J. D. Musa and P. A. Hamilton, Program for software reliability and system test schedule estimation – program documentation, IEEE Computer Society Repository, Ref. No. R277-243.

FAILURES EXPERIENCED VS. EXECUTION TIME

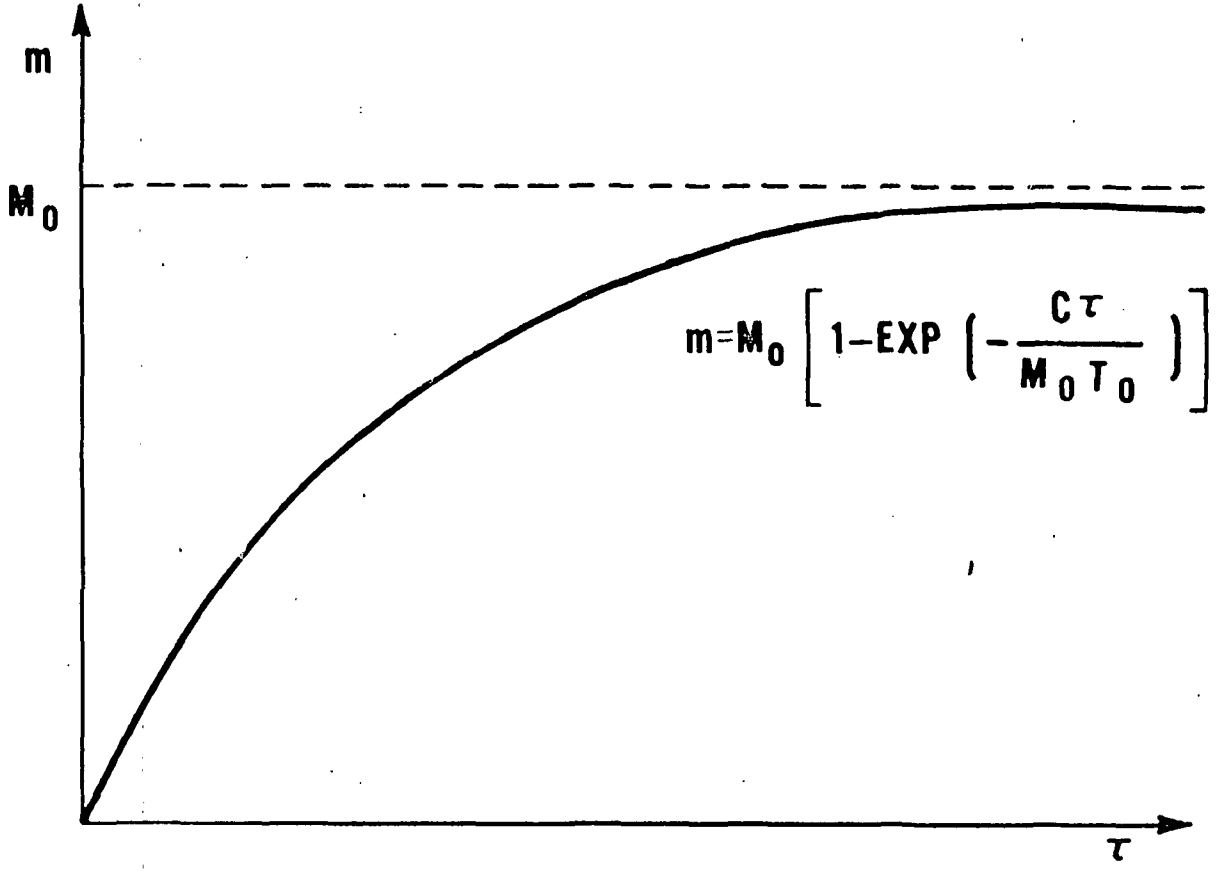


FIGURE 1. EXECUTION TIME MODEL RELATIONSHIP.

J. Musa
Bell Lab.
6 of 11

PRESENT MTTF VS. EXECUTION TIME

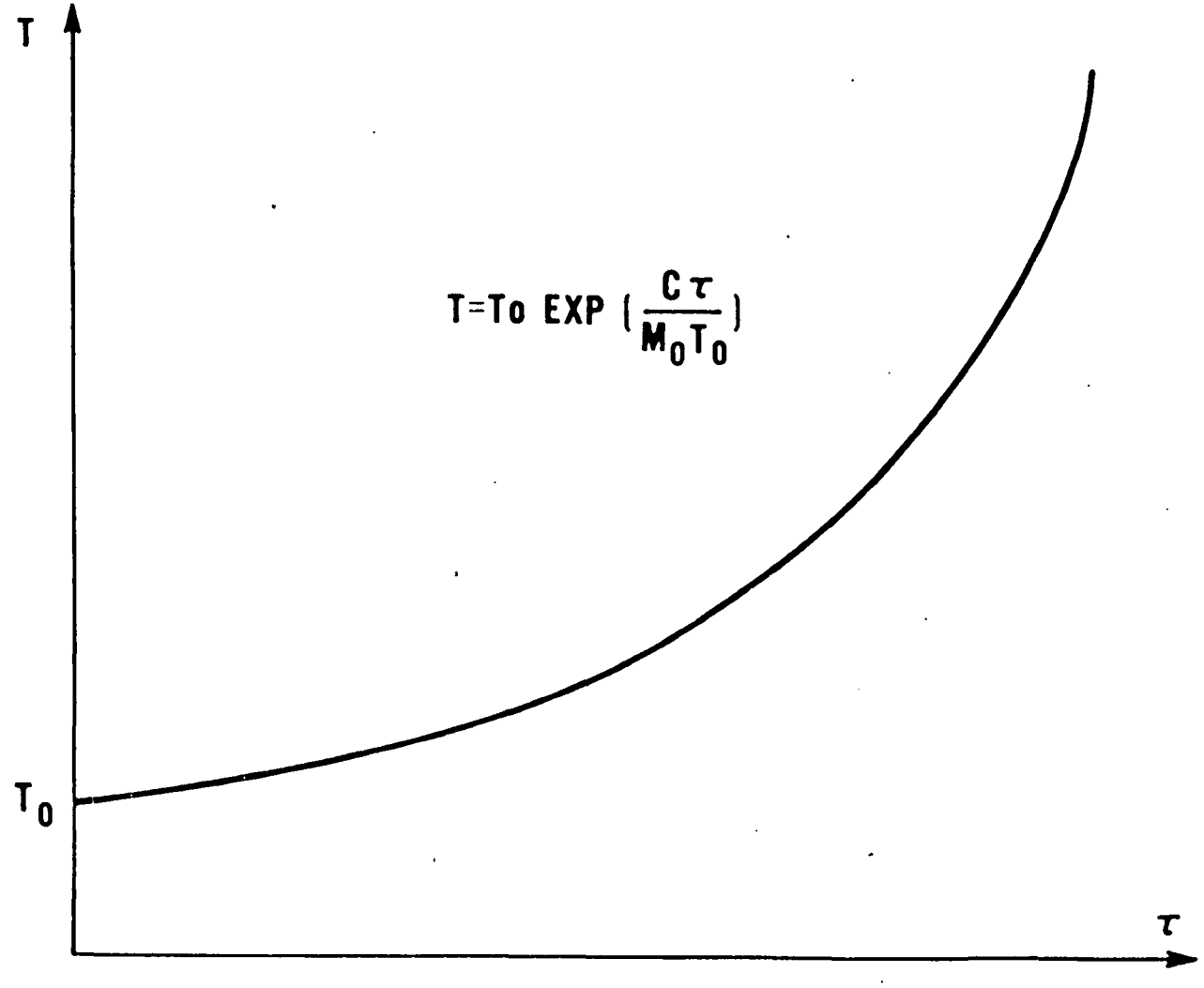


FIGURE 2. EXECUTION TIME MODEL RELATIONSHIP.

J. Musa
Bell Lab.
7 of 11

SOFTWARE RELIABILITY PREDICTION
PROJECT 1

BASED ON SAMPLE OF 136 TEST FAILURES
EXECUTION TIME IS 25.34 HRS
MTTF OBJECTIVE IS 27.80 HOURS
CALENDAR TIME TO DATE IS 96 DAYS
PRESENT DATE:11/ 9/73

	CONF. LIMITS				MOST		CONF. LIMITS		
	95%	90%	75%	50%	LIKELY	50%	75%	90%	95%
TOTAL FAILURES	136	136	136	138	142	148	152	163	182
INITIAL MTTF(HR)	0.522	0.617	0.701	0.744	0.847	0.949	0.992	1.08	1.17
PRESENT MTTF(HR)	999999	999999	999999	30.9	20.4	14.5	12.5	9.53	7.05
PERCENT OF OBJ	100.0	100.0	100.0	100.0	73.4	52.0	45.1	34.3	25.4
*** ADDITIONAL REQUIREMENTS TO MEET MTTF OBJECTIVE ***									
FAILURES	0	0	0	0	2	5	7	12	23
EXEC. TIME(HR)	0	0	0	0	2.46	6.09	7.94	12.4	19.4
CAL. TIME(DAYS)	0	0	0	0	0.958	2.85	4.03	7.39	13.8
COMPLETION DATE READY	110973	110973	110973	110973	111273	111473	111673	112173	112973

Figure 3. Sample Output from Software Reliability Measurement/ Estimation Program for Execution Time Model

SOFTWARE SYSTEM 4

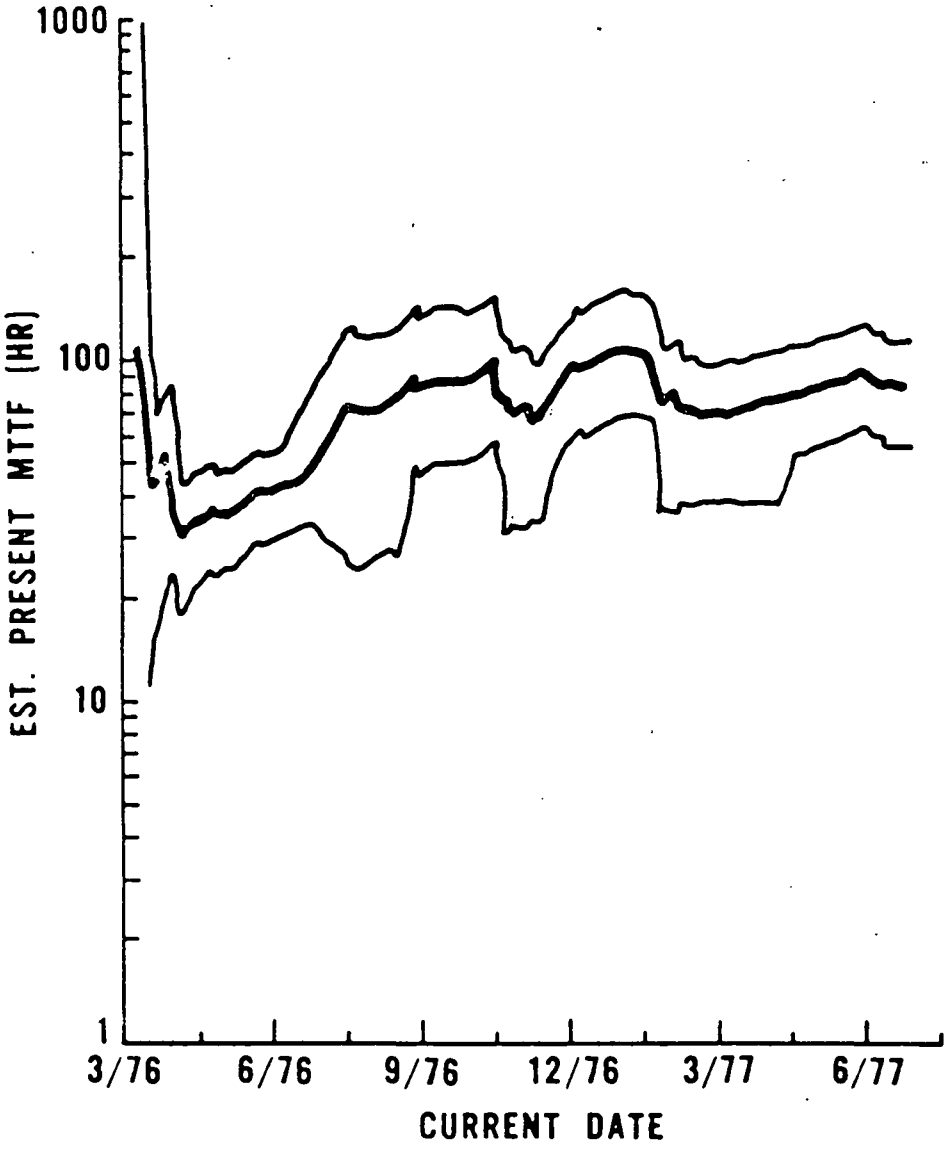


Figure 4. Software Reliability Measurement

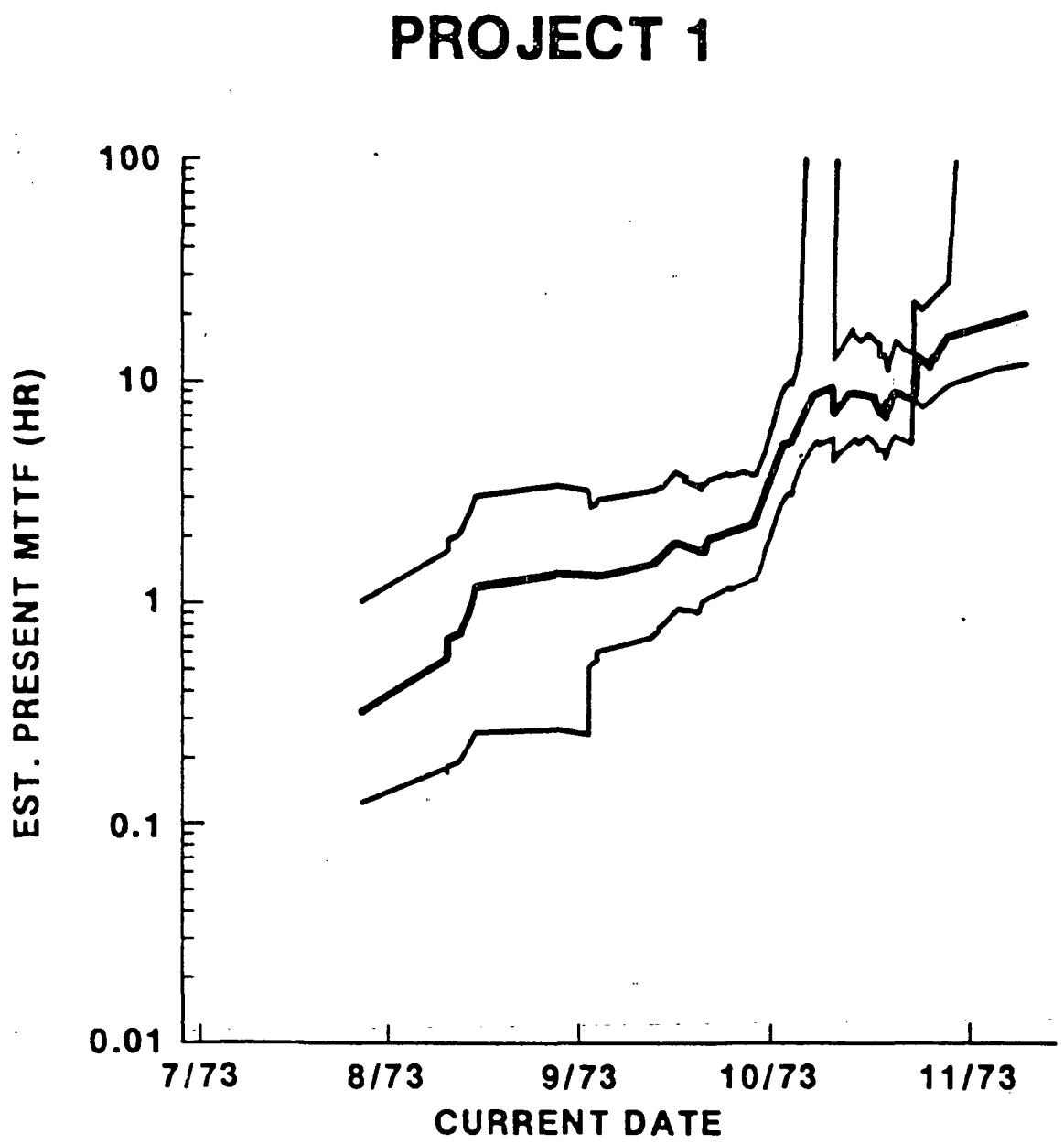
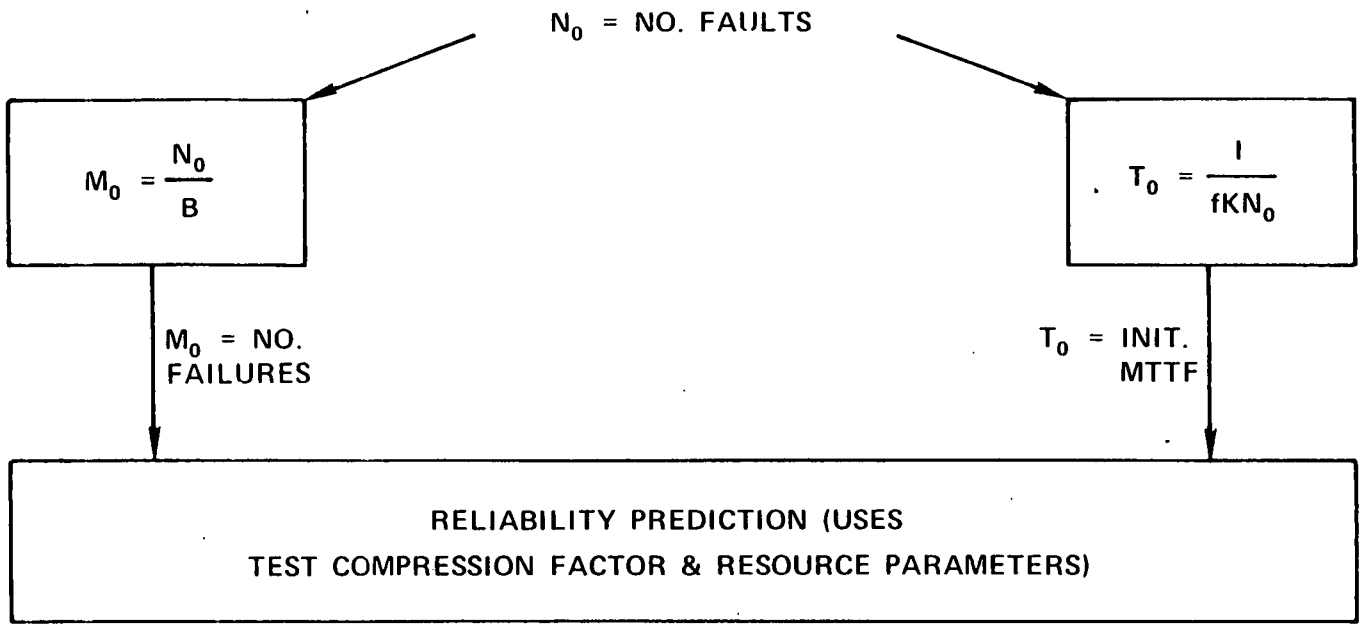


Figure 5. Software Reliability Estimation

SOFTWARE RELIABILITY PREDICTION



B = FAULT REDUCTION FACTOR

f = LINEAR EXEC. FREQ. OF PROGRAM

K = FAULT EXPOSURE RATIO

QUALITY OF INPUTS: BLACK→GOOD,GREEN→SATISFACTORY,RED→FAIR

J. Musa
Bell Lab.
11 of 11.