PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Predicting Software Reliability [and Discussion]

B. Littlewood and A. J. Mayne

Phil. Trans. R. Soc. Lond. A 1989 **327**, 513-527
doi: 10.1098/rsta.1989.0007

Phil. Trans. R. Soc. Lond. A **327**, 513–527 (1989)
Printed in Great Britain

Predicting software reliability

By B. Littlewood
Centre for Software Reliability, City University, Northampton Square, London EC1V 0HB, U.K.

This paper surveys some aspects of the state of the art of software reliability modelling. By far the greatest effort to date has been expended on the problem of assessing and predicting the reliability growth which takes place as faults are found and fixed, so the greater part of the paper addresses this problem. We begin with a simple conceptual model of the software failure process in order to set the scene and motivate the detailed stochastic models which follow. This conceptual model suggests certain minimal characteristics which all growth models for software should possess. There are now several detailed models which aim to represent software reliability growth, but their accuracy of prediction seems to vary greatly from one application to another. As it is not possible to decide *a priori* which will give the most accurate answers for a particular context, the potential user is faced with a dilemma. There seems to be no alternative to analysing the predictive accuracy on the data source under examination and selecting for the current prediction that model which has demonstrated greatest accuracy on earlier predictions for that data. Some ways in which this selection can be effected are described in the paper. It turns out that examination of accuracy of past predictions can be used to improve future predictions by a simple recalibration procedure. Sometimes this technique works dramatically well, and results are shown for some real software failure data. Finally, there is a brief discussion of some wider issues which are not covered by a simple reliability growth study. These include cost modelling, the evaluation of software engineering methodologies, the relationship between testing and reliability, and the important issues of ultra-high reliability and safety-critical systems. On the last point, a warning note is sounded on the wisdom of building systems which depend on software having a very high reliability; this will be very hard to achieve and even harder to demonstrate.

1. Introduction

Estimating and predicting software reliability is not easy. Perhaps the major difficulty is that we are concerned primarily with *design* faults. The situation is very different from that tackled by the conventional hardware reliability theory. Here, the dramatic advances of the past quarter century have resulted from a concentration on the random processes of physical failure. Thus, for example, we now have a good understanding of how the reliabilities of complex hardware systems depend upon, on the one hand, the detailed system structure, and on the other, the reliabilities of the constituent components. The very success of this *physical* hardware reliability theory, however, is now revealing the importance of design faults to the overall reliability of complex systems. Our ability to use intelligent strategies to minimize the effects of physical failure of components results in a higher proportion of system failures being caused by flawed designs.

Software, on the other hand, has no significant physical manifestation. Software failures are merely inherent design faults revealing themselves under appropriate operational

circumstances. These faults will have been resident in the software since their creation in the original design or in subsequent changes.

It seems important to recognize this non-physical nature of the software failure process before attempting to model and predict it. It will be necessary, though, to ensure that the measures of reliability we use for software (and for hardware designs) are compatible with those which have traditionally been used for the physical failure process. After all, the objective is always an understanding of the reliability of a total system, which can suffer software and hardware design failures and physical failures.

One of the reasons why it is becoming vital to have measures of software reliability is the increasing ubiquity of computer systems. It is now common for relatively mundane consumer durables, such as washing machines and video recorders, to have sophisticated computer controls involving quite extensive software. The reliability levels required of these systems are not extremely high, but retro-fitting of modifications can be expensive because of the large numbers of copies.

Purchasers of software today often find that the producers attempt to avoid any responsibility for its incorrect performance. Software warranties, even of a primitive kind, are extremely rare. It is doubtful whether this situation can continue in an increasingly competitive market, and the first vendors of mass-market software to stand behind a clear warranty may gain an important market advantage. Indeed, new U.K. and European consumer protection laws may force vendors to be responsible for users' consequential losses from software failures. In such circumstances it will be important to predict the frequency (and costs) of such events.

In this paper we shall look at some software reliability techniques in detail. These will concern the problem of measuring and predicting software reliability growth during program debugging. Currently this is the most highly developed area and some of these techniques are finding their way into industrial practice (Currit *et al.* 1986). Unfortunately, this is only a small part of the problem. In the final section of the paper, therefore, we shall look at problems for which no current solutions exist. These are generally areas of intensive research, and some attempt will be made to indicate the timescales of likely success. In particular, we shall look closely at the very difficult problem of software in safety-critical systems for which the reliability requirements can be enormously demanding.

2. The software reliability growth problem

2.1. *A conceptual model of the failure process*

Table 1 shows a subset of some software failure data collected several years ago by John Musa at Bell Labs (Musa 1979), and since widely used by workers in software reliability modelling. This particular program was part of a real-time command and control system for which Musa was project manager, which ensured careful data collection. The observations are the execution times, rounded to the nearest second, between successive failures. Although the data were collected during test, the environment was carefully contrived to resemble as closely as possible real operational use. Thus the failure behaviour here should be similar to that which shall be experienced by the user.

As failures occur, attempts are made to fix the underlying faults which they reveal. Musa assumes that each fault is successfully removed before the program is set running again. Notice that this is a questionable assumption in many cases; experience suggests that, not only are fixes 'unsuccessful', but they occasionally cause novel difficulties by inserting new faults.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

TABLE 1. SOFTWARE FAILURE DATA

(Rounded execution times (seconds) between successive failure. Read left to right in rows.)

3	30	113	81	115
9	2	91	112	15
138	50	77	24	108
88	670	120	26	114
325	55	242	68	422
180	10	1146	600	15
36	4	0	8	227
65	176	58	457	300
97	263	452	255	197
193	6	79	816	1351
148	21	233	134	357
193	236	31	369	748
0	232	330	365	1222
543	10	16	529	379
44	129	810	290	300
529	281	160	828	1011
445	296	1755	1064	1783
860	983	707	33	868
724	2323	2930	1461	843
12	261	1800	865	1435
30	143	108	0	3110
1247	943	700	875	245
729	1897	447	386	446
122	990	948	1082	22
75	482	5509	100	10
1071	371	790	6150	3321
1045	648	5485	1160	1864
4116				

A cursory inspection of the raw data shows that the reliability is improving, revealed by the tendency for the inter-failure times to grow larger in the later stages. However, there is great variability and some small inter-failure times are being observed quite late. The several zeroes are rounded down fractions of a second. Musa claims that these features of the data are a natural consequence of the stochastic nature of the failure process and not, for example, a result of bad fixes causing almost immediate failures.

Data of this kind are acquired sequentially as testing progresses. Questions we might ask at each stage of testing include: How reliable is the program now? How reliable will it be at some specified future time? How soon can we expect to achieve a specified target reliability? Before we can answer such questions by using models of the failure process, we need to understand qualitatively what is happening.

We shall take it that the execution of a program involves the selection of inputs from some space I (the totality of all possible inputs), and the transformation of these inputs into outputs (comprising *in toto* a space O). It is worth noting that input spaces are typically extremely large and in most cases a complete description will not be available.

The operational profile of the user will determine the probabilities of selection of different inputs during execution. It is often the case that different users of a program have different operational profiles; our remarks here will address a single such profile.

A program fails when an input is selected which cannot be transformed into an acceptable output. The totality of such inputs we shall call I_F. In practice a failure will be detected by a comparison between the output obtained by processing a particular input, and the output which ought to have been produced according to the specification of the program. Detection

of failures is, of course, a non-trivial task, but we shall not concern ourselves with this problem here.

We can take the conceptual model a stage further by considering the underlying faults which reside in a program p. If we make the reasonable assumption that each failure can be said to have been caused by one (and only one) fault, we have a partitioning of I_F into subsets corresponding to the different faults.

When we successfully remove a fault, and so change the program p into a new program p', the effect is to remove certain points of I from I_F. Thus the members of the removed fault set now map into 'acceptable' regions of O.

Operational use of a program may be thought of as the selection of a trajectory of points in the space I. Typically, many inputs will be successful, i.e. outside I_F, before an input is selected which lies in I_F and so causes a failure. When the failure occurs, an attempt will be made to correct the underlying fault and if this attempt is successful some points are eliminated from I_F. Execution of the program then restarts (most probably in a region outside I_F as I_F is typically very small), and the trajectory of successive inputs continues until the next failure, when the fault is again corrected. The result is a sequence of programs $p_1, p_2, p_3, \ldots$, and a sequence of successively smaller sets $I_{F,1}, I_{F,2}, I_{F,3}, \ldots$. Clearly, the reliability growth is determined by the sequence $\{I_{F,t}\}$.

In this paper we shall confine ourselves to the continuous time case. There is a sense, of course, in which the whole problem is really a discrete time one (computer systems are discrete devices; our conceptual model relies on the idea of discrete input cases). However, the times between successive failures, which will be the random variables of interest, shall typically be very much larger than the machine cycle time and the times required to process individual inputs. Therefore a continuous time approach will be a good approximation to what is really happening.

With this proviso, it seems reasonable to assume that the sets I_F are encountered purely randomly in the execution trajectory. That is, the time to next failure (and so the inter-failure times) has, conditionally, an exponential distribution. If we let $T_1, T_2, \ldots$ be the successive inter-failure times, we have a complete description of the stochastic process if we know the rates $\lambda_1, \lambda_2, \ldots$.

Clearly, these rates, and in particular the successive differences representing the improvements caused by the attempts to remove faults, will depend on the 'sizes' of the subsets representing the faults in I_F. There will be a tendency for the larger faults to be detected, and so removed, earlier; this implies a law of diminishing returns for debugging. However, the failure subset I_F will be encountered randomly and so will its subsets corresponding to the faults. There is no guarantee that faults will be encountered in order of their size. In fact the sequence of successive fault sizes is a stochastic process, and so, therefore, is the corresponding sequence of rates corresponding to $T_1, T_2, \ldots$. Call these rates $\Lambda_1, \Lambda_2, \ldots$.

As this model stands, we would expect that $\Lambda_1 > \Lambda_2 > \ldots$. However, we have so far assumed that a fix is certain to be effective, and therefore the only uncertainty concerns its magnitude. In fact this may be unrealistic. There is evidence that fix attempts are fallible and that sometimes a program is made less reliable as a result of an attempt to remove a fault. It might be more realistic, therefore, merely to insist that the $\{\Lambda_t\}$ sequence is stochastically decreasing.

To summarize this conceptual model so far, there are two sources of uncertainty in the failure behaviour of software which is undergoing debugging. In the first place, there is uncertainty

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

arising from the operational environment, specifically in the sequence of input cases selected for processing by the program. Even if we knew I_F, we should not know when an input would next be selected from here. This uncertainty results in the assumption of conditional exponental distributions for the inter-failure times. Secondly, there is uncertainty arising from the debugging operation itself. Even if we knew the partitioning of I_F, we should not know which fault would be encountered next and so we should be uncertain about the magnitude of the change in the failure rate. This results in the sequence of rates (parameters of the exponential distributions of the Ts) being a stochastic process; that is, we have a doubly stochastic scheme.

2.2. *Some software reliability growth models*

The conceptual model described above is, in some ways, too naïve; it could be seriously misleading for the kind of real-time process control employed in many safety-critical applications. However, even this simple doubly stochastic scheme is not followed in all the models which are found in the literature.

Probably the best known model is one of the earliest. The Jelinski–Moranda (J.–M.) model (Jelinski & Moranda 1972), particularly in its extended form as the Musa model (Musa 1975), is also still one of the most widely used. This model assumes that debugging begins when a program contains N faults, and that each fault contributes the *same* amount, ϕ, to the overall rate of occurrence of failures. Thus, as the fixes which are carried out at failures are assumed to be perfect, the random variables T_1, T_2, ... are independently and exponentially distributed with parameters $N\phi$, $(N-1)\phi$, ... successively.

There are two related criticisms of this model. It treats the sequence of rates as purely deterministic and, more importantly, assumes all faults to have equal size. There is plenty of empirical evidence to the contrary, that faults vary dramatically in their contributions to program unreliability (Nagel & Skrivan 1981; Adams 1984). The reliability predictions obtained from the model are optimistic as a result of this assumption (Littlewood *et al.* 1983).

The Littlewood model (Littlewood 1981) overcomes this difficulty by treating the rates corresponding to the different faults as independent gamma (α, β) random variables. In this model there is a tendency for the larger rate faults to be encountered earlier than the smaller ones, but this sequence is itself random. The model therefore represents the diminishing returns in improved reliability which come from additional testing.

Each of these models is an example of a general class of exponential-order statistic models (Miller 1986*a*). The faults can be seen as 'competing risks'. The times to encounter the different faults are independent, identically distributed random variables (exponential for J.–M., Pareto for Littlewood), and the successive inter-failure times seen by the user are the spacings between the order statistics.

A simple early model which captures the doubly stochastic nature of the conceptual model of the failure process is due to Littlewood & Verrall (1973). Here, the usual assumption is made that the inter-failure times, T_i, are conditionally independent exponentials with rates Λ_i, and the Λ_i are assumed to be gamma $(\alpha, \psi(i))$. Here, $\psi(i)$ is a parametric function which determines the reliability growth (or decay). If $\psi(i)$ is an increasing function of i, it is easy to show that $\{\Lambda_i\}$ is stochastically decreasing and $\{T_i\}$ stochastically increasing. Here we shall use $\psi(i) = \beta_1 + i\beta_2$. Notice that the sign of β_2 then determines whether there is growth or decay in reliability; the data themselves determine whether or not the reliability is increasing.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Non-homogeneous Poisson processes (NHPPs) are obvious candidates for modelling the non-stationary behaviour revealed by data like that in table 1. A minor conceptual drawback is that most such processes have rate functions which change continuously in time; it could be argued that in our case the only changes which take place are the jumps which occur at a fix. However, one way of constructing an NHPP is to assume that N (the total number of initial faults) is Poisson distributed in the exponential-order statistic models such as J.–M. and Littlewood (Miller 1986*a*). Thus the Goel–Okumoto model (Goel & Okumoto 1979) is such an NHPP variant of J.–M. It is easy to show that, on the basis of a single realization, it is not possible to distinguish between such an exponential-order statistic model and its NHPP variant.

Numerous other rate functions have been proposed for NHPP models, including the Duane model (Duane 1964; Crow 1977) which was originally devised for *hardware* reliability growth arising from burn-in testing (the elimination of faulty components in complex systems through their early failure).

These are only a few of the models which have been proposed over the years. Although it is possible to argue that some of these are less plausible than others, we are clearly not in a position to select a definitively 'best' one. As we shall show next, the answers obtained from different models on the same data set can differ dramatically. It turns out, also, that the accuracy of models varies from one data-set to another (Abdel-Ghaly *et al.* 1986; Littlewood 1988). All this suggests that we need a mechanism for selecting among the alternative models for each data-set.

2.3. *Example of use*

The questions we wish to ask about software reliability generally involve *prediction*. Even the simplest question we can ask, concerning the current reliability at a particular stage of debugging, relates to the time-to-next failure random variable and so is a prediction. Users of the models are concerned with obtaining predictions which are sufficiently accurate for their purposes. Selection between the many available models should therefore use predictive accuracy as the primary criterion.

For simplicity, in this section we shall consider only the prediction of the time-to-next failure, i.e. estimation of the current reliability of the program under test. Thus we want to estimate $F_i(t) \equiv P(T_i < t)$, based on the data $t_1, t_2, \ldots, t_i - 1$ previously observed. Clearly, this involves an intermediate step of statistical inference on the unknown parameters, e.g. (N, ϕ) in J.–M. This could involve maximum likelihood estimation of parameters or, more plausibly, a bayesian analysis culminating in bayesian predictive distributions (Aitchison & Dunsmore 1975). The former approach is used here.

Figure 1 summarizes this kind of prediction for various models operating on the data of table 1. Here are shown the medians of the times to next failure estimated by the models on the evidence of the inter-failure times so far observed. This is the kind of repeatedly updated calculation which would be carried out if a user were interested in stopping testing and debugging as soon as a prespecified target reliability had been achieved. Of course the models can present the predictions in other ways (examples are the rate of occurrence of failures and the reliability function). Medians are shown here only for simplicity. Similarly, the models are able (with varying degrees of difficulty) to make longer-term predictions.

At first, the results shown in figure 1 seem disappointing. While the models agree that reliability growth is present, they disagree about the nature and extent of that growth. In

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

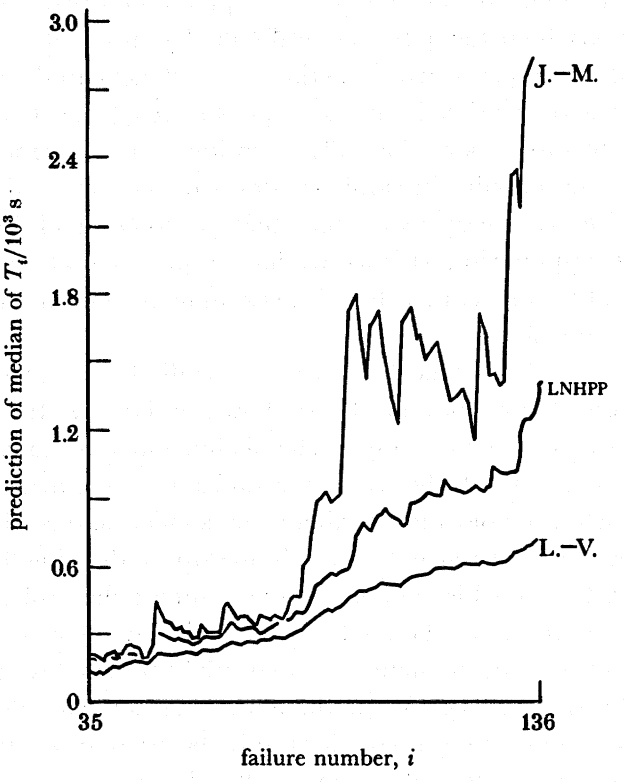

FIGURE 1. One-step-ahead median predictions from the J.–M., LNHPP (Littlewood non-homogenous Poisson process), and L.–V. (Littlewood–Verral) models, using the data of table 1.

particular, some models give a much more optimistic picture in the later stages of debugging than others. It is also notable that some predictions are more 'noisy' than others; these fluctuations might suggest that there are local set-backs occurring in the overall growth in reliability.

A user might reasonably ask which, if any, of these models can be trusted. Unfortunately, this kind of disagreement is typical. Worse, there is evidence (Abdel-Ghaly *et al.* 1986; Littlewood 1988) that the accuracy of the models varies from one data source to another. It is not possible to select a 'universally best' model by comparing performances on many data sets. A user is therefore faced with the problem of deciding which, if any, of the reliability predictions he can trust for this particular data source.

Techniques are now available to detect different kinds of disagreement between predicted and actual failure behaviour, and so assist the user to make a decision. Details are available elsewhere (Abdel-Ghaly *et al.* 1986); here, we shall merely show how a consistent 'bias' in a predictor can be detected and, to a large extent, eliminated.

For simplicity we shall continue to study only the problem of predicting the random variable T_i, having observed $t_1, t_2, \ldots, t_i-1$. We want a good estimate of $F_i(t) \equiv P(T_i < t)$ or, equivalently, of the reliability function $R_i(t) = 1-F_i(t)$. From a particular model we obtain an estimate $\tilde{F}_i(t)$. The user is interested in the closeness of $\tilde{F}_i(t)$ to the unknown true $F_i(t)$. The difficulty is that we never, even after the event, know the true $F_i(t)$; the most we see is a single realization t_i of a random variable with distribution $F_i(t)$.

However, as the user will be making a sequence of predictions $\{F_i(t)\}$ as debugging proceeds, he will gain information about the accuracy of the model on this data source from the pairs $\{\tilde{F}_i(t), t_i\}$. An informal examination will sometimes be sufficient to detect when the sequence $\{t_i\}$ does not look like a realization from the sequence $\{\tilde{F}_i(t)\}$, i.e. that the model is giving inaccurate answers. Slightly more formally, consider the sequence of transformations $u_i = \tilde{F}_i(t_i)$; each u_i is a probability integral transform of the observed t_i using the *previously* calculated prediction, $\tilde{F}_i$. It is easy to see that the u_is should 'look like' a random sample from the uniform distribution $U(0, 1)$ since we have a prequential forecasting system in the spirit of Dawid (1984). One procedure, then, is to examine the sample cumulative distribution function (CDF) of the sequence of us.

Two such u-plots, for the J.–M. and Littlewood–Verrall (L.–V.) models, each making 100 predictions using the data source of table 1, are shown in figure 2. It can be seen that L.–V. is closer to the uniform line of unit slope. The Kolmogorov distances are 0.190 (J.–M.), significant at the 1% level, and 0.144 (L.–V.), significant at the 5% level. More important than this, however, is what the plots tell us about the detailed nature of the prediction errors. The plot for J.–M. is everywhere above the line of unit slope, indicating that there are too many small u values. That is, the model is tending to underestimate the probability of failure before t_i (the later observed failure time); the model is too optimistic. Conversely, there is evidence that L.–V. is too pessimistic in its predictions. A user might reasonably conclude that the truth lies somewhere between L.–V. and J.–M. predictions. Thus, for example, in figure 1 we might expect the Littlewood NHPP (LNHPP) predictions to be better than L.–V. and J.–M. In fact a u-plot of these predictions is close to uniform (the Kolmogorov distance, 0.098, is not significant).

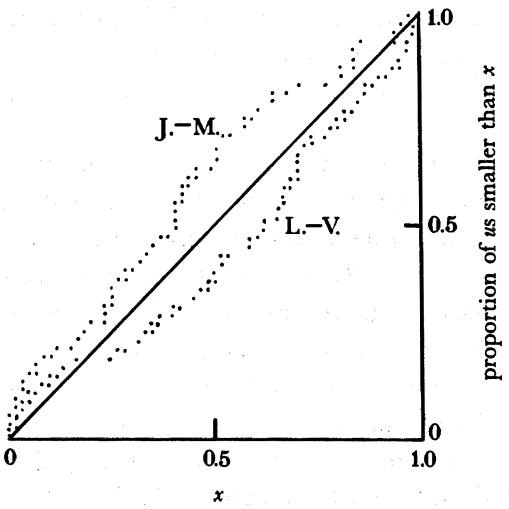

FIGURE 2. u-plots for the predictions of next inter-failure time from the J.–M. and L.–V. models, using the data from table 1.

A comparison of the performance of ten software reliability models on this data, using u-plots and other criteria (Abdel-Ghaly *et al.* 1986), shows that in fact the LNHPP is giving the best results overall. In the absence of any other information, a user might conclude that, for *future* predictions on the same data source, this would be the preferred choice.

2.4. *Recalibration*

The ability of the *u*-plot to give some indication of the nature of the deviation between predicted and actual failure behaviour ('pessimism', 'optimism') suggests that it might be possible to improve the raw predictions emanating from a model. Formally, there must exist functions G_i such that $F_i(t) = G_i[\tilde{F}_i(t)]$. If the relation between predicted and actual behaviour is approximately stationary, i.e. G_i is approximately independent of i, it may be estimated from past predictions. In fact the joined-up *u*-plot (Keiller & Littlewood 1984), or some suitably smoothed version of this (Chan *et al.* 1985), will serve as such an estimate.

In order to recalibrate the raw prediction, $\tilde{F}_i(t)$, at stage i, based on $t_1, t_2, \ldots, t_{i-1}$, the procedure is as follows.

1. Check that the relationship between $\{\tilde{F}_i(t)\}$ and $\{F_i(t)\}$ is approximately stationary (see, for example, Littlewood 1988 for a simple procedure).
2. Find the *u*-plot for predictions made *before* stage i, i.e. based on subsets of $t_1, t_2, \ldots, t_{i-1}$. Join up the vertices and then, if necessary, smooth the resulting polygon (Chan *et al.* 1985). Call the resulting function G_i^*.
3. Calculate the raw prediction, $\tilde{F}_i(t)$.
4. Recalibrate to obtain $\tilde{F}_i^*(t) \equiv G_i^*[\tilde{F}_i(t)]$.

This procedure is repeated at each stage i. Notice the sequence $\{\tilde{F}_i^*(t)\}$ comprises genuine predictions; each $\tilde{F}_i^*$ is obtained using only information observed earlier. Thus we can examine their accuracy using exactly the same procedures as adopted for $\{\tilde{F}_i(t)\}$. For example, for the data of table 1, the Kolmogorov distances of the *u**-plots are (J.–M.) 0.104 and (L.–V.) 0.084, compared with 0.190 and 0.144 respectively. (Note that these are based on 80 predictions, not the 100 in the earlier results, because the first recalibration is based on the first 20 raw predictions). This improvement is shown clearly in the medians in figure 3; J.–M.* is less optimistic than the originally too optimistic J.–M., L.–V.* is less pessimistic than the two pessimistic L.–V.

Recent simulation results show that recalibration works well in surprisingly wide circumstances (Brocklehurst 1987). For example, even when there is evidence of non-stationarity in the relationship between predicted and actual behaviour. Occasionally the results are dramatic. Figure 4 shows orders-of-magnitude disagreement between raw predictions, and all 10 models applied to this data gave very poor results; recalibration brings models into much closer agreement.

2.5. *Summary*

The results here and elsewhere (Abdel-Ghaly *et al.* 1986; Littlewood 1988) show that it is possible to investigate the accuracy of the predictions of software reliability growth emanating from several models on a single data source. Of course, there is no guarantee that a model which has given accurate predictions in the past for a particular data source will necessarily do so in the future. In the case of software it is particularly important to ensure that the conditions of use remain the same for the period of data collection and the period for which predictions are to be made. This can be difficult when it is desired to predict user-perceived reliability on the basis of vendor test data. It is necessary to create a testing régime which accurately represents the operational use of the program; see Currit *et al.* (1986) for a successful application in an industrial context.

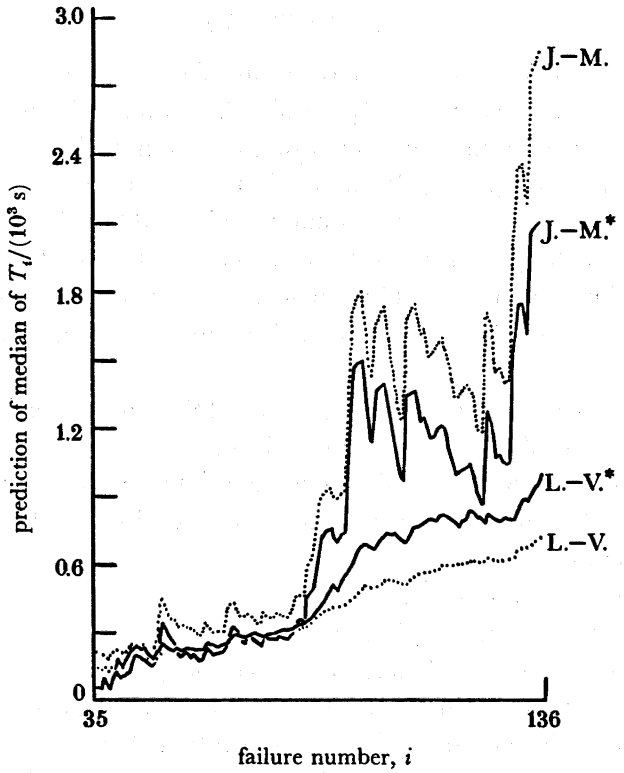

FIGURE 3. Effect of adaptive procedure on the J.–M. and L.–V. predictions for the data of table 1.

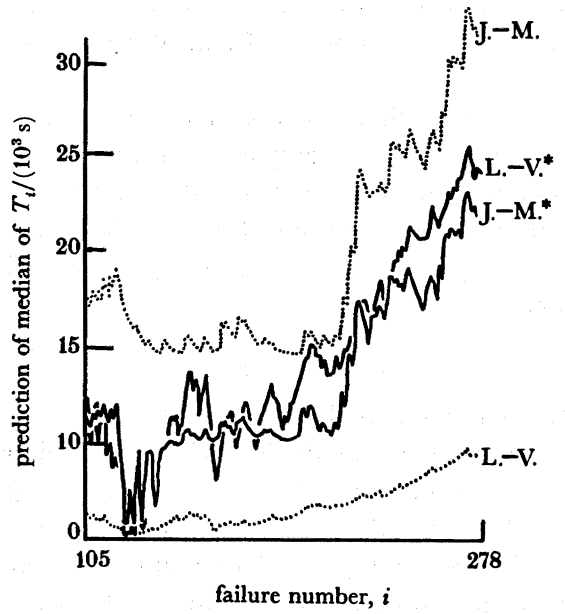

FIGURE 4. Successive median predictions using Musa's data (1979). Notice the way the adaptive procedure brings two sets of predictions into close agreement when the raw predictions differ dramatically.

With these caveats, it does seem that it is possible in most cases to obtain reasonably accurate reliability predictions for software and, more importantly, have confidence in their accuracy even though no single model can be universally trusted.

The simple recalibration technique described here generally works well. Its performance does not have to be taken on trust, however; because it forms a genuine prequential forecasting system when it operates on a raw model, all the usual analytic techniques are available for examining predictive accuracy.

Having said this, it has to be admitted that the reliability growth described here forms only a small aspect of the reliability problem facing software engineering. In the next section we shall look at other aspects of the problem. This is an active research area and it is likely that the next few years will see important advances as a result of the several coordinated information technology research programmes (e.g. Alvey in the U.K. and the European Strategic Programme for Research in Information Technology). There may remain, though, intractable problems in some particularly important areas.

3. Wider aspects of software reliability

The problem of measuring and predicting reliability cannot really be separated from the methodology for the attaining of reliability. The latter can range from the very important management issues associated with software development to problems of testing methodology, verification, fault-tolerant architectures and metrication of the process. Neither should the emphasis in this paper on software issues allow us to forget that the software will always be merely a part of an overall system; it is the performance of this system which matters to the user. Wider issues like these cannot be discussed here for reasons of brevity. In what follows we shall consider a few software reliability problems selected for their interest and potential importance.

3.1. *Cost models*

Reliability models almost invariably only consider the failure events, and do not treat the consequences of failure. Users, on the other hand, usually have in mind the fact that failures are going to cost something. Recent consumer-protection legislation may make vendors liable for the consequential losses suffered by users. If this occurs, it will be necessary to be able to predict the cost process in order to have a rational pricing policy and possibly to fix insurance premiums. Such problems are already encountered in simpler form by vendors who need to estimate the costs of field maintenance of software. Cost models do not seem to be intrinsically difficult extensions of some reliability models, and this is an active research area. However, there do seem to be difficulties in obtaining data on the costs themselves. This is an area where industry could usefully ease the constraints of commercial confidentiality and share some information (for example, by means of the U.K. Alvey Software Data Library).

3.2. *Software testing*

It has been indicated earlier that it is difficult to construct testing régimes which emulate operational environments, and so allow estimates to be made of user-perceived reliability. It is also often argued that such testing is inefficient as a means of removing bugs and hence achieving reliability. More conventional test strategies, it is claimed, allow the tester to use knowledge about the likely types of faults present in order to remove them more efficiently.

Such techniques do not, however, produce data which allow the reliability of the program under test to be evaluated.

There is thus an apparent conflict between testing seen as a means of achieving reliability, and testing seen as a means of evaluation. We need to reconcile these conflicting aims by developing new testing strategies.

3.3. *Evaluation of development process and techniques*

The cost-effective delivery of reliability is an important criterion against which competing software development methodologies can be judged. The goal would be the provision of means whereby project managers could make informed decisions about the best techniques to be used in particular contexts.

Consider, for example, fault tolerance and testing. Under what circumstances is it best to spend heavily on the testing of a single version of a program? There is evidence that in some cases it may be more cost-effective to build a fault-tolerant system, but we are currently not in a position to resolve this choice by taking into account particular circumstances.

Proportional hazard modelling (Kalbfleisch & Prentice 1980) has been suggested as a method of identifying the relative importance of factors which influence ultimate product reliability. The difficulty is that the experimental unit here will be a complete development of a program, with a history of reliability growth such as described in §2 as well as explanatory variables related to process and product. Any analysis would require many such experimental units. Once again, the availability of suitable data has been a stumbling block to all except the most naïve investigation along these lines.

3.4. *Safety-critical systems*

An important problem here is the achievement and assurance of very high reliability. For example, it has been stated (Rouquet & Traverse 1986) that the fly-by-wire computer system for the Airbus A320 has a reliability requirement of a failure rate of 10^{-9} h^{-1}, as loss of function cannot be tolerated.

Clearly, the reliability growth techniques of §2 are useless in the face of such ultra-high reliability requirements. It is easy to see that, even in the unlikely event that the system had achieved such a reliability, we could not assure ourselves of that achievement in an acceptable time. Mathematical verification techniques may eventually become available for these large control programs, but they cannot address the problem of faults in the specification. To know that the program is formally 'correct' is to know that it correctly implements a formal specification, not that this specification accurately represents requirements (e.g. for safety). Users of these systems (and their nature is often such that we are all 'users') rightly expect an assurance that they shall fail acceptably infrequently.

Design diversity for fault tolerance has been advocated as a means of achieving high reliabilities cost-effectively. Several systems have been built using such techniques (including that of the Airbus A320 (Rouquet & Traverse 1986)). Unfortunately, there is evidence that independently developed software versions will not fail independently, and so will not deliver the dramatic increases in reliability over single versions which a naïve assumption of independence would suggest. Knight & Leveson (1986), for example, report an experiment in which 27 versions were developed independently but contained a high incidence of common faults. Eckhardt & Lee (1985) give a theoretical scenario, based on the notion of varying 'difficulty' of different inputs, which supports these empirical findings.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

These problems make it unlikely that in the foreseeable future we shall be able to use design diversity to achieve ultra-high reliability. More importantly, they render the assurance problem essentially impossible. Even in the unlikely event of our having successfully built a system with such an extremely high reliability, we would never be able to assure ourselves or others of this success. We cannot appeal to simple hardware-like theories of redundancy with independent failures, but must try to estimate the dependence between versions. This has been shown to be as difficult as the problem of simply testing the complete fault-tolerant system as a black box (Miller 1986*b*), and so is essentially impossible.

This discussion is not meant to imply that these techniques will not be found to be useful. Both formal verification and fault-tolerance methods are likely to become important tools for software engineers. They are clearly not, at this stage, a solution to the problem of both achieving and assuring ultra-high reliability.

It is surprising to find that systems are being built whose safe functioning relies upon software having these unassurable reliability requirements.

4. Conclusion

There is a great need to measure software reliability in a wide range of contexts. Our ability to satisfy this need, however, varies greatly. On the positive side, it is now possible in many cases to obtain accurate reliability estimates from appropriate debugging data. Furthermore, it is usually also possible in such cases to actually analyse the accuracy of the estimates and predictions, so that there is no need to appeal to previous experience of a 'good' model.

There are, however, rather stringent requirements on the testing environment for these techniques to be successful. Most importantly, it should accurately resemble the actual use environment if we wish to predict user-perceived reliability. Also, these techniques are practical only for relatively modest reliability levels; for ultra-high reliability it would be necessary to observe the system's failure behaviour for several orders-of-magnitude longer than likely system lifetimes.

There are several areas of active research in software reliability. The most promising are concerned with cost models; characterization of the user's operational environments so that reliability can be matched *a priori* to type of use; relationship between reliability measurement and conventional testing strategies. Surprisingly, in view of the great attention it has attracted, the problem of using properties of the product and its development process to improve reliability prediction seems more problematical. Even 'complexity', which many agree militates against reliability, is not at present captured in a measure which has found universal acceptance; much less do we understand its effect on a product's failure behaviour.

The problem of ultra-high reliability remains doubly intractable; there exist neither proven methods of achieving it nor methods of measuring it in a particular context. Like difficult problems in mathematics, however, it inspires interesting new results (formal verification and fault-tolerant techniques are two). But these should be seen at present as potentially cost-effective ways of achieving relatively mundane reliability levels. Until we can assure ultra-high software reliability, perhaps we ought to curb the dependence of safety-critical systems on the failure-free operation of computer programs.

It is appropriate to end with a plea for better and more plentiful data sources to further knowledge in this area. This remains a difficulty for several reasons. Most companies are reluctant to share data for reasons of commercial confidentiality; it is often felt that reliability

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

data reflects upon the quality of company products and so is particularly sensitive. From a statistical viewpoint, the experimental unit here is a single program and its development history. Each observation is thus extremely expensive and replication has so far been rare. One solution is to study relatively small and unrealistic programs in an academic context. A more promising avenue might be international cooperation on a more true-to-life experimental programme. This would extend and enhance the experimental work which is now almost exclusively confined to the United States.

This work was supported partly by a grant from NASA Langley Research Center, U.S.A. and partly by a grant from the Alvey Directorate and SERC, U.K.

References

Abdel-Ghaly, A. A., Chan, P. Y. & Littlewood, B. 1986 Evaluation of competing software reliability predictions. *IEEE Trans. Software Engng* **SE-12**, 950–967.

Adams, E. N. 1984 Optimizing preventive service of software products. *IBM J. Res. Dev.* **28** (1), 2–14.

Aitchison, J. & Dunsmore, I. R. 1975 *Statistical prediction analysis*. Cambridge University Press.

Brocklehurst, S. 1987 On the effectiveness of adaptive software reliability modelling. *CSR Tech. Rep.* Oct. 1987. London: City University.

Chan, P. Y., Littlewood, B. & Snell, J. 1985 Parametric spline approach to adaptive reliability modelling. *CSR Tech. Rep.* July 1985. London: City University.

Crow, L. H. 1977 Confidence interval procedures for reliability growth analysis. *Tech. Rep.* **197**. Aberdeen, Maryland: U.S. Army Material Systems Analysis Activity.

Currit, P. A., Dyer, M. & Mills, H. D. 1986 Certifying the reliability of software. *IEEE Trans. Software Engng* **SE-12**, 3–11.

Dawid, A. P. 1984 Statistical theory: the prequential approach. *Jl R. statist. Soc.* A **147**, 278–292.

Duane, J. T. 1964 Learning curve approach to reliability monitoring. *IEEE Trans. Aerospace* **2**, 563–566.

Eckhardt, D. E. & Lee, L. D. 1985 A theoretical basis for the analysis of multi-version software subject to coincident errors. *IEEE Trans. Software Engng* **SE-12**, 1511–1517.

Goel, A. L. & Okumoto, K. 1979 Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. Reliability* **R-28**, 206–211.

Jelinski, Z. & Moranda, P. B. 1972 Software reliability research. In *Statistical Computer Performance Evaluation* (ed. W. Freiberger), pp. 465–484. New York: Academic Press.

Kalbfleisch, J. D. & Prentice, R. L. 1980 *The statistical analysis of failure time data*. New York: Wiley.

Keiller, P. A. & Littlewood, B. 1984 Adaptive software reliability modelling. In *Digest 14th International Symposium on Fault-tolerant Computing*, pp. 108–113. New York: IEEE Press.

Knight, J. C. & Leveson, N. G. 1986 An empirical study of failure probabilities in multi-version software. In *Digest 16th International Symposium on Fault-tolerant Computing*, pp. 165–170. New York: IEEE Press.

Littlewood, B. 1981 Stochastic reliability growth: a model for fault removal in computer programs and hardware designs. *IEEE Trans. Reliability* **R-30**, 313–320.

Littlewood, B. 1988 Forecasting software reliability. In *Bayesian Methods in Reliability* (ed. P. Sander). Dordrecht: Reidel. (To appear).

Littlewood, B., Keiller, P. A., Miller, D. R. & Sofer, A. 1983 On the quality of software reliability predictions. In *Proc. of NATO ASI on Electronic Systems Effectiveness and Life Cycle Costing* (ed. J. Skwirzinski), pp. 441–460. Heidelberg: Springer-Verlag.

Littlewood, B. & Verrall, J. L. 1973 A Bayesian reliability growth model for computer software. *Jl R. statist. Soc.* C **22**, 332–346.

Miller, D. R. 1986*a* Exponential order statistic models of software reliability growth. *IEEE Trans. Software Engng* **SE-12**, 12–24.

Miller, D. R. 1986*b* Making statistical inferences about software reliability. In *Newsletter of Alvey Software Reliability and Metrics Club*, no. 4. London: CSR, City University.

Musa, J. 1975 A theory of software reliability and its application. *IEEE Trans. Software Engng* **SE-1**, 312–327.

Musa, J. 1979 Software reliability data. *Tech. Rep.* (Available from Data Analysis Center for Software, Rome Air Development Center, New York, U.S.A.).

Nagel, P. M. & Skrivan, J. A. 1981 Software reliability: repetitive run experimentation and modelling. *Tech. Rep.* BCS-40399. Seattle, Washington: Boeing Computer Services Company.

Rouquet, J. C. & Traverse, P. J. 1986 Safe and reliable computing on board the Airbus and ATR aircraft. In *Proc. of Fifth IFAC Workshop on Safety of Computer Control Systems* (ed. W. J. Quirk), pp. 93–97. Oxford: Pergamon Press.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Discussion

A. J. Mayne (*Milton Keynes, U.K.*). I am glad that Professor Littlewood emphasized the importance of the human factor in software reliability. In practice, what is under test is not an isolated software system, but a software system plus a human–software interface. This interface includes software documentation and perceptions and misperceptions of the software.

Another way in which the human factor may affect software reliability is through patterns of input of specific parameter values by particular types of users. For example, software that has been substantially 'debugged' may have errors that are activated by only a small subset of input-parameter combinations. Then it is possible for the software to run smoothly for a long time after its presumed debugging, just because its users so far had not inputted any of its critical parameter combinations.

A third way in which the human factor should be taken into account is when a programmer's attempt to correct a software error introduces further errors. As far as I know, few if any software reliability models have taken much account of this effect.

Another area connected with software reliability is the corruption of files stored on magnetic discs and other media. File corruption of this sort can on occasion give rise to faults which are then falsely attributed to software unreliability. This situation can occur in at least two ways. Occasionally, files can be corrupted or made 'inaccessible' as a result of obscure hardware faults. More commonly, in situations where the same data file may be accessed by several different programs, an error in one of these programs sometimes corrupts the file in such a way that the other programs cannot retrieve or process their data properly.

B. Littlewood. Mr Mayne is quite right to point out that it can be very misleading to talk of *the* reliability of a program when that program may be executed in many different operational environments. In fact, all our probability statements should specify which operational environment (or mode of use) applies.

The problem of characterizing operational environments (essentially measuring their 'stressfulness') is an interesting one which invites study. Ideally, we would like to test a program in one environment and be able to predict its reliability in a totally different new environment. Intuitively, it seems this may be possible; the difficulty is the one of combining the failure data from test with appropriate measures of the characteristics of the novel environment.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES