**Edward N. Adams**

# Optimizing Preventive Service of Software Products

*The implementer of a large, complex software system cannot make it completely defect free, so he must normally provide fixes for defects found after the code is put into service. A system user may do preventive service by installing these fixes before the defects cause him problems. Preventive service can benefit both the software developer and the software user to the extent that it reduces the number of operational problems caused by software errors, but it requires the expenditure of the resources required to prepare, disseminate, and install fixes; and it can be the cause of additional software problems caused by design errors introduced into the code by fixes. The benefit from removing a given defect depends on how many problems it would otherwise cause. Benefits may be estimated by modeling problem occurrence as a random process in execution time governed by a distribution of characteristic rates. It is found that most of the benefit to be realized by preventive service comes from removing a relatively small number of high-rate defects that are found early in the service life of the code. For the typical user corrective service would seem preferable to preventive service as a way of dealing with most defects found after code has had some hundreds of months of usage.*

## Introduction

It is difficult to create a very large software product that is completely free of design errors (DEs). Accordingly, the software producer must provide means to correct user code for DEs not eliminated during product development. We term the process of correcting a user's code to eliminate a DE that is causing him a problem Corrective Service (CS) and the process of correcting his code to eliminate a DE that has not yet caused him a problem Preventive Service (PS).

Where a software system has many hundreds or thousands of users it is commonly the case that for every user problem caused by a previously unknown DE (a "discovery"), there will be several problems caused by already known DEs ("rediscoveries"). In such a situation, if all users would perform the PS to remove a DE soon after it is discovered, it would be possible to avert most of the user problems requiring CS to DEs in the product software. A program of thorough PS could be of considerable value to the software producer, since to the extent that it reduced his software product costs, it would permit him to realize higher effective product quality and to offer service at a lower price; and it could benefit the software user both in terms of lower cost and higher quality of software and in terms of a reduction in costs due to unscheduled interruptions of service caused by product software DEs.

However, PS like CS has its costs. It consumes the developer resources needed to prepare and distribute media for mass dissemination of the fixes, and it consumes user resources of operational time and staff to install fixes preventively. Moreover, PS is to some extent an original source of problems, since the code fixes themselves occasionally inject new DEs into the code.

The studies reported here were done over the period 1975–80 with the objective of developing means to estimate whether and under what circumstances PS is worthwhile to do. We report on

- The model we used to project the occurrence of user software problems;
- The results of fitting the model to historical error data;
- Our conclusions about how to carry out an optimum program of PS.

## The scenario for software service

Software service of the products we studied involved this series of events: The user of an IBM software product reported to an IBM Customer Engineer (CE) that he had a problem believed to be caused by a DE in the product. The CE diagnosed the source of the problem; if it was a DE in IBM code, he reported it to an IBM change team in a form called an APAR. The change team studied the APAR and, if the DE proved valid, prepared and sent to the user a code change that would avert the user's immediate problem.

Next the change team refined and tested the code change more comprehensively, and incorporated the finished version into a Program Temporary Fix (PTF), which is a vehicle for transmitting the changed module(s) of the product code. The change team transferred the PTF to another IBM group, which assembled PTFs on tapes for periodic distribution to users. Later an IBM development group might further process the PTFs and incorporate them into an updated version of the product. An IBM library facility distributed PTF tapes at frequent intervals, approximately monthly in the first months after First Customer Shipment (FCS) of the product. A PTF tape typically incorporated fixes for all significant errors found in the product up to a time one or two months before the tape was put in the library, and it took about a month to get the tapes from the distribution center to the average user. Accordingly, a user having the most recent PTF tape could assume that he had fixes for most errors discovered more than three months earlier.

Some users installed all PTFs preventively, some installed only selected PTFs, some just kept PTFs as an immediate source for fixes if need arose. Some users would install the PTFs only after waiting to see whether other users who had installed them had encountered new DEs caused by the fixes in the PTFs.

More often than not when a DE caused a user problem the problem proved to be a rediscovery. A rediscovery of a DE was somewhat easier to deal with than an original discovery, since much was already known about the DE and fixes for it were available. Even so a rediscovery was costly both to IBM and to the user.

For many products having large usership, problems caused by rediscoveries were more numerous than problems caused by original discoveries, even to the extent that they were the dominant factor in service cost. For such products the IBM service organization encouraged users to do prompt PS for all DEs. However, the users seemed more aware of the costs and risks of PS than they were persuaded of the benefits. (It is difficult to demonstrate the benefits realized from PS, since they must be quantified in terms of hypothetical events that PS is believed to have averted.) For whatever reason most users were unwilling to preventively install PTFs *en masse*.

## Modeling the occurrence of errors

How to evaluate the benefits from PS is simple in principle. Some interval of time is required after a DE is discovered to prepare a change to the base code and install it in the users' versions of the code. Rediscoveries that occur in this time interval cannot be avoided; rediscoveries that would have occurred after this interval will be averted. One must determine the time interval between discovery and fix for the system arrangements in use and project what problems would have occurred after this interval. One then reckons what it is worth to avert these problems.

To carry through such an evaluation one needs means to project the numbers and times of the rediscoveries that would have followed each discovery. When we began our work little was known about the statistics of occurrence of rediscoveries, since no records were kept that summarized for a given DE the rediscoveries that followed it.

However, we had one piece of qualitative information that provided the key to understanding the time patterns of rediscovery. Persons familiar with the service scene reported that particular DEs were quite "virulent," causing problems to many users, but that many DEs were rather innocuous. We realized that if the "virulence" of a DE were a manifestation of the average rate at which the DE caused problems under operational circumstances, the times of rediscovery of a DE could be projected from its quantified virulence.

In accordance with this idea we adopted the following model for dealing with problem occurrence: In operational circumstances each DE in a product manifests a characteristic problem rate $R$. The occurrence of each problem caused by the DE, whether the original discovery or a rediscovery, is an event in a Poisson process, in which problem events occur at random times with an average rate $R$. The cumulative running time of all users plays the role of time in the Poisson process. The error rate of a product is the sum of the error rates associated with the individual DEs. The rates associated with different DEs may range over a wide spectrum of values, but the rate associated with a given DE remains unchanged until it is removed from the code by installation of a fix.

On this model the pattern in real time of the occurrence of software problems for a product is primarily a function of

- The rate of usership vs time,
- The number of DEs originally present in the product,
- The distribution of problem rates over the original DEs, and
- The schedule of installing fixes for DEs after they are discovered.

We sought historical error data for some older products in order to test whether we could estimate virulences and correlate them to patterns of rediscoveries. We soon learned of the work of Richard W. Phillips of IBM Poughkeepsie, who had collected and analyzed data on both product usage and times of original discovery for a number of products of interest to us. Phillips had determined the distribution of original discoveries against cumulative usage for these products and had found that the distribution was similar from one product to another. He had even determined an empirical formula for this distribution.

We found that Phillips' formula for the distribution of discoveries vs usage could be reproduced using our model if we assumed that the rate distribution, i.e., the relative numbers of DEs having each possible rate, was proportional to a particular inverse power of the rates. This finding confirmed the apparent usefulness of our model, so we used it in all our subsequent work.

Phillips was kind enough to make all of his data available to us so we began by studying them in detail. As we did so we saw that we would need a great deal of other data before we could establish whether discoveries and rediscoveries could be successfully projected.

## Methods of data analysis and model calibration

We collected or created the following error statistics to use in applying our model to the analysis of error data for a number of software products:

- The average monthly product usage beginning with FCS,
- The number of original discoveries each month,
- The number of rediscoveries each month of each known DE.

The products we studied were all products such as operating systems comprised of hundreds of thousands of lines of code and used by many hundreds of users.

- *Creating usage statistics*

We needed to accurately determine the relation between usage and time for our products. The rate of usage by a user is his machine speed, and the total rate of usage at any time is the number of user machine speeds, so the cumulative usage $U$ is the integral of

$$dU = N(T) \cdot F \cdot dT,$$

where $N(T)$ is the number of users at time $T$ and $F$ is the average machine speed.

Conceptually the machine speed value for determining usage would depend on the size of a user machine and the average number of shifts per day of usage. In practice we had no direct access to such detailed information about usage; the most we could hope to do was determine what user machines were running in a given month. The usage statistics available to us initially were indirectly arrived at estimates of the numbers of user machines at the end of each calendar quarter. These statistics proved to be insufficiently accurate for our purposes, so we developed our own usage estimates by analyzing records of service to software installed in IBM machines that were under contract for software service. The machines for which detailed records of software service were available to us were those in the United States inventory, so we limited our detailed study of error phenomena to users of these machines who bought software service from IBM. The U.S. inventory of machines accounted for about half of the total usage of our products. We reckoned usage in usage-months, a usage month being a chronological month multiplied by a machine speed.

To formulate usage month statistics we created for each machine a month by month record of which software products were serviced during a given month. We assumed that a product was in use for any month in which it required service, and during all months between two months in which it required service. When competing products such as different operating systems or different versions of the same operating system were in use during the same month—a common situation—we allocated the usage for that month among the pieces of competing software on the basis of the apparent level of service currently being required for each, and where successive versions of a product were in overlapping use, we phased out the older one in a timely fashion. In this manner we built for every machine in the U.S. inventory a putative month-by-month history of product use in which each machine contributed just one month of usage per month to the total usage statistics for all functionally similar products.

Some survey data were available from which we could determine for some products what the total usage had been at particular months. Our estimates checked well at times both early and late in product life.

- *Taking account of machine size*

We assumed originally that the machine size factor $F$ in the usage reckoning should be the machine speed, and we adopted as a measure of machine speed a composite of cycle speed and memory size that had been developed in IBM. However, when we examined machine by machine error data for any single product, we found that problem rates for the same sized machines differed greatly and showed no tendency to distribute about a mean; typically a few machines had many problems, many machines had only a few problems. One obvious source of variation was that a machine on three-shift operation got credit for the same usage as a machine on one-shift operation, but the variations were much

greater than this mechanism could account for. When we learned that some of our users rented many machines, used the same software on each, and assigned to a single machine the task of reporting service problems for all, so that that one machine might be reporting the usage from dozens or hundreds of shifts a day, we realized that we were too far removed from actual operations to understand the error statistics for individual machines.

However, we needed a machine speed calibration. Phillips' discovery that product discovery curves were similar had led to a desire for a planning tool that could forecast the distribution in time of future problem occurrences for a product while it was still in development. For this purpose we determined a relation of the $F$ factor in usage to machine speed that would make the rate distributions determined from different products as similar as possible. Assuming that $F$ varied as a simple power of machine speed, we did a linear regression of data from a number of products to determine the best fit of that power. The resulting power was close to 0.5. We do not assign much meaning to the 0.5, but in our subsequent fittings of product data we used it so that our results were consistent with those of others.

• *Creating discovery statistics*

The problem rates associated with known DEs can only be known approximately and were estimated by statistical inference from the statistics of DE discovery and problem occurrence. For determining discovery times we had available APAR records, which contained the actual date of discovery of each DE. However, many of the APARs were submitted by users not in our user set, either users in other countries or users in internal IBM development groups. Since such APARs did not result from usage included in our usage statistics, it would be incorrect to count them as discoveries in our rate inference calculations. Accordingly we took as the date of DE discovery for our purposes the date of the first discovery by one of our users and created a statistical series "Discoveries by U.S. field users under service contract."

We also made minor adjustments to the discovery statistics of products for which there were successive releases in which a successor release contained essentially all of the code of the previous release. The adjustments in such cases were to count a few DEs found in successor releases as original in the earlier release, even though not found there before the release of the new code. We do not discuss the procedure for making these adjustments, which in any case affected only DEs having problem rates so low that they could remain undetected until release of successor software.

• *Creating problem statistics*

Our most accurate means of estimating the problem rates associated with a DE requires us to know the total number of problems, both original discovery and rediscoveries of the DE in a known period of usage. Such statistics had not been collected before our work, but fortunately there was a set of records from which the information to construct them could be extracted, viz., the weekly effort reports of the persons who service the products, the individual IBM Customer Engineers (CEs). Each CE prepared a weekly report about problems worked on that week. Each problem worked on was assigned a unique code number, which it kept for as long as the problem was open. The weekly report recorded the code number of each problem worked on during the week, what was done in regard to it, and, if the cause of a problem was finally diagnosed as an IBM DE, the APAR number of the DE causing the problem.

Gordon Jones and his associates in the IBM group involved in our task saw the possibility to determine rediscovery times by analyzing the CE effort reports. They developed a set of programs that searched the archival file of activity reports, pieced together all references to each problem number worked on, determined when each problem was closed for the last time, identified the APAR number of the DE that caused the problem, and determined whether the problem involved an original discovery or a rediscovery. Their programs built problem files from which we could determine for each DE in our products the date of the original discovery and the dates of each subsequent rediscovery.

• *Estimating problem rates associated with a DE*

We used two methods for estimating the rate associated with a DE. Each method gave a distribution of probabilities that the DE had each possible rate $R$.

*Method 1: estimation from usage at time of discovery*

On our model the probability that a given DE will cause a problem in usage interval $dU$ is

$$R\,dU,$$

where $R$ is the problem rate associated with the DE. The probability that a DE present in the code at FCS would remain there undetected after usage $U$ is

$$e^{-RU},$$

and the probability that a DE having rate $R$ would survive usage $U$ without causing a user problem and be discovered for the first time in usage interval $dU$ is

$$R \cdot e^{-RU}\,dU.$$

By Bayes' Theorem the probability that a given DE discovered for the first time at $U$ has rate $R$ is

$$\frac{P(R,0)\,R\,e^{-RU}\,dU}{\sum_{R'} P(R',0)\,R'\,e^{-R'U}\,dU},$$

in which $P(R, 0)$ is the probability for a DE to have rate $R$ if we had no information about it.

The derivation of this formula may be seen as follows: The DE that caused the problem at $U$ in $dU$ must have some one of the possible rates. The numerator is the probability that the DE had the rate $R$ and that it caused a problem at $U$ in $dU$. The denominator is the sum of the independent probabilities that the DE had one of the possible rates and caused a problem at $U$ in $dU$. The denominator, since it includes all possible ways for the event to occur, is the probability that the DE would be found at $U$ in $dU$; so the fraction is the probability that the discovery came about in $dU$ at $U$ through the case in which DE has rate $R$.

If we set the factor $P(R, 0)$ equal to 1, as though all values of $R$ are equally probable *a priori*, we get the probability density that the rate is in $dR$ at $R$:

$$U^2 R e^{-RU}.$$

This function of $R$ has a maximum at $R = 1/U$. The rate estimated from the maximum is just the reciprocal accumulated usage at the time the DE is discovered. The expected value of $R$ is $2/U$, twice the value at maximum, which indicates that the distribution is skewed far toward higher values.

*Method 2: estimation from usage at time of N problems*
The probability that a DE having rate $R$ will cause exactly $N$ problems during cumulative usage $U$ is

$$\frac{(UR)^N e^{-RU}}{N!}.$$

By the same kind of reasoning as for method 1, the probability that a DE found exactly $N$ times during usage $U$ has associated rate $R$ is

$$\frac{(RU)^N e^{-RU} P(R, 0)\, dU}{\sum_{R'} (R'U)^N e^{-RU} P(R', 0)\, dU}.$$

If we take all $R$ values as equally likely *a priori*, we find the probability density that the rate is in $dR$ at $R$ to be

$$\frac{U(RU)^N e^{-RU}}{N!}.$$

This function of $R$ has its maximum at $R = N/U$ and expected value $(N + 1)/U$, which is $(N + 1)/N$ times the value at maximum. If $N$ is large, the skew toward large values is much smaller percentagewise than in the distribution obtained by method 1.

- *Comparison of the two methods*

When many discoveries of a DE have occurred, the estimate of method 2 will have a much lower variance than that of method 1 and is quite insensitive to the initial probability factors in the statistical formulas. For DEs having multiple discoveries it is presumably the more accurate estimate provided that the usage interval of analysis is one in which a negligible amount of PS was done.

If many users have removed a DE from their code, method 2 will give values of $R$ that are too small in proportion as the usage by users who actually had the DE in their code is smaller than the total usage, since it counts the usage of those users who have actually stopped having the problems associated with these DEs. The uncertainty of estimate because of possible PS begins to be significant when the problem data are based on a period ending more than a few months after the time of discovery, and grows progressively greater the longer the time from discovery to the time of estimate. We show below data that provide evidence of the effect of PS on rates estimated by method 2.

The estimate by method 1 is not affected directly by the amount of PS done, but it is affected indirectly. If the estimate is made after much PS has been done, some of the discovered DEs will actually be DEs introduced by fixes. These secondary DEs will be assigned rates that are too low, and sometimes very much too low; and they will be used to impute low-rate DEs to the original distribution that were not actually there. To estimate the significance of this effect we need an estimate of how many secondary DEs are present. We report below a study we did with such a model, in which we found that the total number of originally present latent DEs estimated from late life data were about 15 percent higher when we did not take account of the creation of secondary DEs than when we did. We may take this as an estimate of the extent to which the original number of DEs of low rate is overestimated by method 1. Method 1 should not systematically overestimate or underestimate the number of DEs of high problem rate; however, its estimates of these numbers is quite sensitive to the initial probability factors $P(R, 0)$ in the formulas of statistical inference, so its results are subject to the uncertainties involved in assigning these factors.

- *Creating an empirical rate distribution*

We created an empirical rate distribution by adding together the rate distributions from the individual DEs found. For most purposes we estimated original rate distributions using method 2 and usage periods less than a year. We could estimate a rate distribution from the set of problems found in any continuous usage interval beginning with FCS. To do so we went through the following steps:

- We constructed a one-rowed table giving the numbers of DEs for which there were 1, 2, 3, · · · problems in that interval.

- We determined for each DE in the table the probability for its rate to take each value $R$ of the distribution and assigned to each rate of the distribution a fractional number of DEs equal to that probability.
- We totaled up the contributions of fractional DEs for each value of $R$ to get the estimated rate distribution for the DEs that had been found.
- We adjusted this distribution by dividing by

  $1 - e^{-RU}$,

  which is the fraction of DEs of rate $R$ originally present that would have been expected to be found in usage $U$. The adjusted distribution is our estimate of the rate distribution of DEs in the code as originally shipped.

A basic practical question was how to represent the distribution we were to fit. Given a known theoretical form for the distribution, we might have parameterized that form and chosen the parameters to attain a best fit. Having no established theory to provide such a form, we chose to derive a purely empirical distribution. A completely arbitrary distribution would have required an infinite number of parameters of fit, so we made the assumption that the distribution was smooth, so that if fitted at a number of points it could be inferred in between.

The empirical distribution we used was a set of discrete points, spaced at logarithmic intervals so that each rate above the lowest was larger than the one below it by a factor square root of 10. This distribution provided a point fairly close (within about a half of a Naperian interval) to any possible rate value in the range of values.

We experimented with a number of different choices of the discrete rate values and also studied fitting with different numbers of discrete values. We found that in general we required a range of several orders of magnitude in problem rates to represent the error patterns at all well; a minimum number of six rate values seemed needed and seven or eight seemed slightly better in some cases.

In practice we chose to parameterize the rate distribution in terms of *percentages* of DEs associated with each rate value plus the estimated total number of DEs present at FCS, hoping to be able to correlate the total number with the number of lines of new code.

- *Choosing the prior probability factors*

The rate distributions derived in the manner described are sensitive in part to the choice of the initial probability factors $P(R, 0)$. Unfortunately, there is no uniquely correct way to assign these factors. The results that are sensitive to these factors are rates assigned to a DE for which the problem count is low; the rates assigned to a DE having only one problem are essentially determined by the prior probability factors.

When we began our work we chose these factors by assuming that all rates for a DE were equally likely *a priori*. To get a feeling about the underlying distribution we then performed this process:

- Determine a distribution from the data,
- Use the distribution arrived at as a new trial distribution to provide values for the $P(R, 0)$,
- Repeat until the distribution converges.

We would expect this procedure to converge to a distribution for which the actual data are more probable than any other data. While there is no reason to expect that the actual rate distribution would have that property, we expected it to be one for which the data are probable, so hoped the one converged on might be similar. Later, when we had distributions from a number of products and verified from Phillips' finding that they were similar, we felt it reasonable to use the average of several distributions to estimate the factors $P(R, 0)$.

In most of our work we used rates estimated by method 2 with problem data from the early field use of the code. These data include few discoveries of DEs having very low problem rates and no rediscoveries of them, so could give little empirical information about the true rates for these DEs. Since these DEs are by far the most numerous in the code, the *total number of original latent DEs* may be estimated very wrongly. Accordingly, we also made some determinations of the rate distribution using method 1 with discoveries over most of the usage life of the code. This analysis is known to overestimate the number of low-rate DEs, but because of our study of secondary DEs we felt that its value for the total number of latent DEs originally present might not be in error by a large factor.

Finally, by choosing our prior probability factors so that the two kinds of estimate of the rate distribution were fairly consistent, we sought to protect ourselves against getting these factors grossly wrong.

- *A study of secondary DEs*

All attempts to fit our simple model rate distribution to empirical problem data over the life of the code showed a systematic error of fit: The model with the best fitted parameters would give more early discoveries and fewer late discoveries than were actually observed. We thought that this failure of the modeling was probably caused at least in part by the fact that some of the later-found DEs had been injected into the code via the fixes for earlier-found DEs, as we could see from direct evidence.

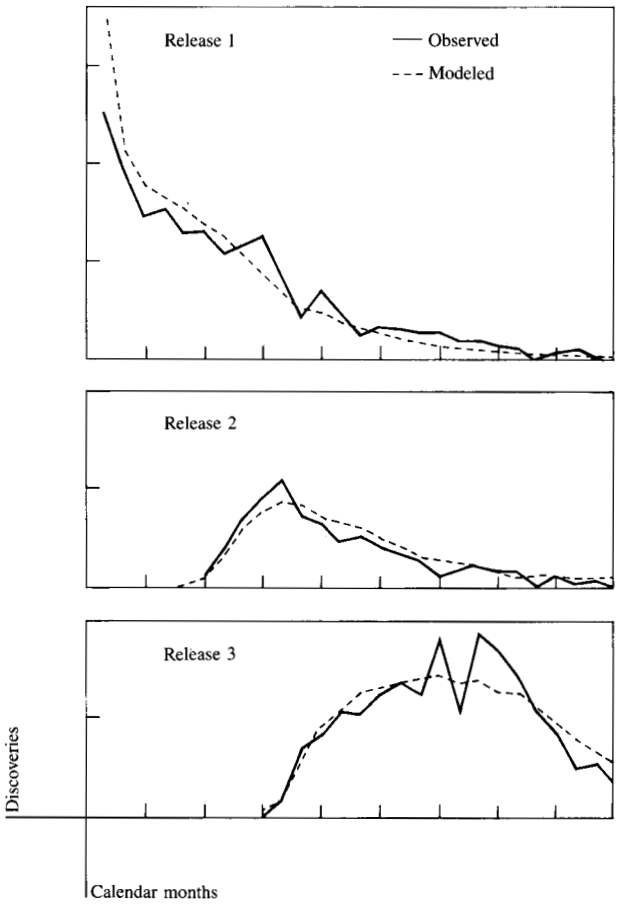


**Figure 1** Monthly discoveries for products of Table 1. The actual and projected plots of discoveries vs time for three successive releases of a product. The discoveries are projected from the fitted numbers of original DEs for each release (not shown) and the mean of the three rate distributions.

We wanted to estimate approximately how much error the presence of secondary DEs would cause in estimating the parameters of the original rate distribution using an analysis that ignored these DEs. To this end we made a modified model that incorporated a representation of the introduction of secondary DEs as PS is done. This modified model was based on a number of overly simple assumptions, e.g., that a constant fraction of fixes introduced new DEs, that the new DEs had the same distribution of rates as newly shipped code, that the same constant fraction of users did PS at all times in the life of the code, that PS was done with a constant time lag after initial discovery, etc.

Since the treatment of PS was so artificial, we were not concerned to improve the fitting of empirical data in general, but only to determine whether we could find plausible values of fit parameters that would eliminate the systematic imbalance between early- and late-found DEs. We found that we could roughly account for the excess of long term discoveries if we assumed that for each fix installed a new DE was generated with probability of the order of fifteen percent. Unfortunately the modified model involves so much that is arbitrary and overly simple that a detailed report on it would involve more explanation than the results justify. Apart from its value to give us an indication of how important secondary DEs were, it was of no greater value than the unmodified model.

## Fitting the model parameters to some service histories

Having developed the sources of data and methods of analysis described above, we analyzed service data for a number of IBM products. Because of the commercial significance of the data and other sensitivities, we do not present original data nor identify individual products. We can characterize the products: Each was a release of an operating system, a large component of such a release, or a product of similar character in size and manner of use.

- *Fitted rate distributions*

Table 1 shows the fitted rate distribution for three successive releases of a single operating system. The reciprocal rate values, i.e., the mean times between user problems, are shown vs the percentage of DEs having this value. Here, as in all the data we present, we show only relative distributions.

- *Fitted total numbers of DEs present*

We do not present data on the estimated number of latent DEs present, since we did not find a way to get consistent and meaningful figures for this number. The difficulty we had is related to our inability to accurately determine the rates of DEs having very low problem rates. By varying the range of rates assumed possibly to be present and by varying the choice of the initial probability factors $P(R, 0)$, we could produce sets of fitting parameters that had widely different numbers of latent DEs but fit the problem data about equally well. These various fitted distributions varied almost entirely in the large numbers of DEs having very low problem rates; they all had about the same numbers of latent DEs present having high problem rates, these being the ones that accounted for the observed problem activity.

We had originally chosen to use the total number of latent DEs as a parameter, hoping to correlate it to the number of lines of new code in the product. Indeed, for a given set of rate parameters we did find that the number of latent DEs was crudely proportional to the number of lines of new code, but with deviations from proportionality of a factor of 2. However, it is clear that most of the fitted latent DEs did not correspond to real DEs of which our data could give knowledge. So the fitted total number of latent DEs is merely a fitting parameter whose values are to be understood only in connection with all other parameters used in a particular fit.

### *Fitted discovery distributions*

Figure 1 compares actual discoveries to fitted discoveries vs real time for each of the three releases of Table 1. Each fitted curve was projected from the model using as input data the monthly usage data for the release, the inferred total number of original latent DEs for the release, and the average of the three fitted rate distributions as given in the table. Since each of the three different distributions of discoveries vs real time is obtained from the same average rate distribution, it is clear that the main differences of shape derive from differences in the time pattern of usership. That the shapes are rendered fairly well in such a plot was Phillips' discovery.

Figure 1 shows that our model can capture the principal information needed to understand the smoothed trend of discoveries in real time. Although the time trends of fit are perhaps a little better than typical in this particular case, one can see that at some points there are errors of fit of the order of 30–50 percent, which is what one must expect when modeling in terms of such gross aggregates.

### *Fitting rediscovery distributions*

We did several studies of detailed rediscovery distributions. The format of such studies was to seek any plausible assumptions about what PS might have been done that would permit the model to reproduce the general trend of the rediscovery data. By using the model with secondary DEs we could almost always do such a fit within the accuracy that is to be expected for this kind of model. However, we do not present data showing these fits, since they are based on such arbitrary assumptions about PS that they are persuasive of little.

### *Similarity of rate distributions of different products*

Table 2 tabulates the distributions obtained from fitting problem data for the nine large products for which we had full life data at the time of the analysis. Among these were products used on both large and small machines. (It may be noted that the rate values used here are somewhat different from those used in Table 1. The differences are not important: Either set of values gives about as good a fit of the data.)

Examining the fitted rate data one can see that the numbers in the leftmost columns—the percentage of DEs in the lowest rate values—are very similar from product to product. These similarities are not meaningful, but merely reflect similarities of the initial probabilities used in the fitting as discussed above.

One can see that in the rightmost columns the variations along each column, from column to column, and from product to product also show similarities, although with much more variability of the data. These similarities are meaningful, and the discrepancies between corresponding numbers in these columns are probably indicative of the actual variability of the rate distributions from product to product. Noisy these data are, but they suggest that rate distributions have considerable similarity of form from product to product.

Using the mean of these nine distributions, we calculated the total problem activity contributed by the DEs in each rate group. The result was that the few DEs in the set having the highest rate were responsible for more total problem rate

**Table 1** Problem rates for three releases of a product. The fitted distribution of problem rates is shown as a mean time between problem occurrences vs the percent of DEs in the corresponding rate class.

| | *Rate class* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | *Mean time to problem occurrence in kmonths for rate class* | | | | | | | |
| | 95 | 30 | 9 | 3 | 0.9 | 0.3 | 0.09 | 0.03 |
| | *Fitted percentage defects in rate class by release* | | | | | | | |
| Release | | | | | | | | |
| 1 | 24.1 | 26.4 | 28.8 | 13.5 | 3.0 | 2.9 | 1.1 | 0.2 |
| 2 | 21.3 | 23.2 | 25.5 | 13.8 | 5.0 | 7.4 | 3.3 | 0.5 |
| 3 | 21.9 | 24.1 | 27.5 | 16.2 | 3.9 | 4.0 | 2.1 | 0.2 |
| Average | 22.4 | 24.6 | 27.3 | 14.5 | 4.0 | 4.8 | 2.2 | 0.3 |

**Table 2** Rate distributions for nine software products. These rate distributions were fitted using the same fit program and the same prior probability factors. The square root of machine speed rather than machine speed itself was used for reckoning usage of machines.

| | *Rate class* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | *Mean time to problem occurrence in kmonths for rate class* | | | | | | | |
| | 60 | 19 | 6 | 1.9 | 0.6 | 0.19 | 0.06 | 0.019 |
| | *Fitted percentage defects in rate class by release* | | | | | | | |
| Product | | | | | | | | |
| 1 | 34.2 | 28.8 | 17.8 | 10.3 | 5.0 | 2.1 | 1.2 | 0.7 |
| 2 | 34.3 | 28.0 | 18.2 | 9.7 | 4.5 | 3.2 | 1.5 | 0.7 |
| 3 | 33.7 | 28.5 | 18.0 | 8.7 | 6.5 | 2.8 | 1.4 | 0.4 |
| 4 | 34.2 | 28.5 | 18.7 | 11.9 | 4.4 | 2.0 | 0.3 | 0.1 |
| 5 | 34.2 | 28.5 | 18.4 | 9.4 | 4.4 | 2.9 | 1.4 | 0.7 |
| 6 | 32.0 | 28.2 | 20.1 | 11.5 | 5.0 | 2.1 | 0.8 | 0.3 |
| 7 | 34.0 | 28.5 | 18.5 | 9.9 | 4.5 | 2.7 | 1.4 | 0.6 |
| 8 | 31.9 | 27.1 | 18.4 | 11.1 | 6.5 | 2.7 | 1.4 | 1.1 |
| 9 | 31.2 | 27.6 | 20.4 | 12.8 | 5.6 | 1.9 | 0.5 | 0.0 |

**Table 3** Variation of fitted rates with usage interval. The rate distribution for this product was fitted using the total usage and cumulative discoveries as of the end of each month for 18 months. The apparent decreases in the right-hand columns show that the fraction of users having all of the most virulent DEs in their code diminishes after about month 7 or 8.

| | *Percentage of defects by rate class* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *Class* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| *Month* | | | | | | | | |
| 1 | 23.1 | 24.7 | 26.5 | 14.6 | 4.4 | 4.5 | 2.0 | 0.3 |
| 2 | 23.1 | 24.8 | 26.5 | 14.6 | 4.3 | 4.4 | 1.9 | 0.3 |
| 3 | 23.2 | 24.9 | 26.6 | 14.6 | 4.3 | 4.3 | 1.8 | 0.3 |
| 4 | 23.2 | 24.9 | 26.6 | 14.5 | 4.3 | 4.3 | 1.9 | 0.4 |
| 5 | 23.2 | 24.9 | 26.6 | 14.6 | 4.3 | 4.3 | 1.6 | 0.5 |
| 6 | 23.3 | 25.0 | 26.8 | 14.8 | 4.4 | 4.0 | 1.4 | 0.3 |
| 7 | 23.3 | 25.0 | 26.8 | 14.8 | 4.4 | 4.0 | 1.3 | 0.3 |
| 8 | 23.3 | 25.0 | 27.0 | 15.1 | 4.5 | 3.8 | 1.1 | 0.3 |
| 9 | 22.9 | 24.7 | 26.8 | 15.4 | 5.0 | 3.9 | 1.1 | 0.2 |
| 10 | 23.4 | 25.1 | 27.0 | 15.1 | 4.7 | 3.7 | 0.9 | 0.1 |
| 11 | 23.3 | 25.0 | 26.9 | 15.3 | 5.2 | 3.4 | 0.7 | 0.0 |
| 12 | 23.7 | 25.4 | 27.1 | 14.8 | 5.1 | 3.2 | 0.6 | 0.1 |
| 13 | 23.9 | 25.6 | 27.0 | 14.7 | 5.4 | 2.9 | 0.5 | 0.1 |
| 14 | 24.4 | 25.9 | 26.9 | 14.4 | 5.4 | 2.6 | 0.4 | 0.1 |
| 15 | 24.6 | 26.1 | 27.0 | 14.2 | 5.3 | 2.5 | 0.3 | 0.1 |
| 16 | 24.9 | 26.3 | 27.0 | 14.0 | 5.2 | 2.2 | 0.3 | 0.1 |
| 17 | 25.2 | 26.5 | 26.8 | 14.0 | 5.1 | 2.0 | 0.3 | 0.1 |
| 18 | 25.4 | 26.7 | 26.9 | 13.8 | 5.0 | 1.9 | 0.3 | 0.1 |

than those in the set having the second highest rate, and that in general the DEs in any set were responsible for more problem activity than those in the set having the next highest rate. Thus the small numbers of DEs in the sets having the very highest rates account for nearly all of the problem potential of the code.

• *The effects of PS on fitted rates*

Table 3 illustrates how some effects of PS show up when we analyze data from progressively longer service intervals. The figure shows a table of the rate distribution obtained by fitting the cumulative problem data for a single product after each calendar month for 18 months. The inferred distributions are rather stable for the first seven or eight months, with no more than a suggestion of depletion of the percentages in the highest rate values, but after that time the numbers of DEs assigned to the three highest rate values diminish markedly.

That the numbers in the right-hand columns are fairly stable for the early months implies that as the user months accumulate, the numbers of problems assigned to the most virulent DEs increase in the same proportion, as though most of the users continue to have these DEs in their code.

After about seven months a change occurs, so that the numbers of DEs having high rates diminish markedly. This implies that the total number of problems reported against these DEs no longer increases in the same proportion as the usage, as though many of the users no longer have those virulent DEs in their code. Without looking at the raw data one cannot tell how completely the most virulent DEs had been eliminated, but it is as though their problem activity had been eliminated from most of the code.

We interpret these data to mean that during the first half year after FCS little if any PS was done, since the first DEs that PS would eliminate continued to be present essentially unchanged. We used a table such as this to estimate the amount of PS being done so that we could decide how many months of data could safely be used to determine an original rate distribution.

## Discussion of the rate distribution

Several features of the distributions are of interest. First, the rate distributions are generically similar from product to product, as Phillips had discovered. To the extent that they are similar, we may conjecture that some process has shaped them so. If so, it should be a pervasive process, since these products had been produced at different places and times, by different people, and with significant differences of programming technology.

Second, the approximately linear slopes of the distributions at high rates are less steep than the simple model would suggest. If one considers that during the latter months of development the code is subjected to considerable usage, one might expect the numbers of the most virulent DEs to be reduced far below what is found, in fact that they would be approximately in the ratios of the exponential attenuation factors defined by their problem rates.

Reflecting on these matters and being aware that in fact considerable numbers of DEs are found during the development process, we realized that many must also be added during development. We developed an intuition that the characteristic shape of these distributions might just reflect the balancing during the development process between acts of creation and removal of high-rate DEs. Assume that as development proceeds new DEs are continually created, but at the same time existing ones are found and removed, and imagine that one can represent the evolution of the rate distribution by a pseudo-differential equation

$$\frac{d\,P(R)}{d\,t} = G - R_0\,P(R) - RP(R),$$

where $P(R)$ is the number of DEs per unit interval of rate $R$ and "$t$" is a variable representing the passage of development "time." The term $G$ represents the effective rate at which new DEs are created per unit of $t$; the term with $R$ represents the probable fact that developers find DEs by activities similar in effect to running the code in the field; the term

with $R_0$ reflects the assumption that developers also find DEs in ways that are unrelated to running it in the field. (We have evidence that they do, which need not be reviewed here.)

If development goes on long enough for the process described to come into balance for those DEs having very high problem rates, one may set the left-hand side to zero and solve for $P(R)$:

$$P(R) = \frac{G}{R+R_0}.$$

Since $G$ itself could be expected to depend on $R$ (it is plausible that the developer is more likely to create DEs of low virulence than DEs of high virulence), we cannot conclude the exact form by which the distribution should vary with $R$. However, these considerations suggest why $P(R)$ might well vary inversely with $R$ somewhat as observed. So that we could relate this formula to the actual mean rate distribution of Table 2, we made a best fit of that distribution assuming $P(R)$ to be proportional to a power of $R + R_0$. To do this we associated with each discrete value except the lowest all the rates of the continuous distribution in the logarithmic interval from a 1/4 root of 10 below to 1/4 root of 10 above that discrete value. We associated with the lowest value all the states from rate 0 up to a factor 1/4 root of 10 above the lowest value. We obtained a best fit distribution

$$P(R) = \frac{G}{(R + 0.032)^{1.69}}$$

that fits the mean rate distribution closely. (There is a small glitch at the lowest rate point, undoubtedly related to the extra weight given by integrating the function all the way from 0.)

## Optimizing preventive service

The rough level of modeling we have achieved is adequate as a guide to planning effective PS. We take it as axiomatic that PS is not to be done unless its benefits justify its cost. Generally speaking, the service costs that can be profitably avoided by PS are the result of problems caused by a small number of highly virulent DEs, most of which are found very soon after the code is put into service. The main question about any DE is whether to deal with it by PS or by CS.

The considerations in answering this question are much the same for the product producer and the product user. The user who has had no problem does not want to install a PTF if the risk from possible new DEs plus the costs of doing PS outweigh the expected gains from avoiding problems from the known DE. The producer must prepare a fix for the DE in any case, since he must provide CS to the user who discovered the DE, but he does not want to incur the additional expense of working up PTFs and distributing them to users unless the users will judge it worthwhile to put them in and will avert service calls by doing so.

Discovery month

| Rediscovery month | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 26 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 59 | 35 | 19 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 73 | 43 | 24 | 15 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 88 | 52 | 28 | 17 | 12 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 103 | 61 | 33 | 20 | 14 | 10 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 117 | 69 | 38 | 23 | 16 | 11 | 8 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 132 | 78 | 42 | 26 | 18 | 13 | 9 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 147 | 87 | 47 | 29 | 20 | 14 | 10 | 8 | 6 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | 162 | 95 | 52 | 32 | 22 | 15 | 11 | 9 | 7 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 176 | 104 | 57 | 35 | 24 | 17 | 12 | 9 | 7 | 6 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 191 | 113 | 61 | 38 | 25 | 18 | 13 | 10 | 8 | 6 | 5 | 4 | 2 | 0 | 0 | 0 | 0 | 0 |
| 14 | 206 | 121 | 66 | 41 | 27 | 19 | 14 | 11 | 8 | 7 | 5 | 4 | 3 | 1 | 0 | 0 | 0 | 0 |
| 15 | 220 | 130 | 71 | 44 | 29 | 21 | 15 | 12 | 9 | 7 | 6 | 5 | 4 | 3 | 1 | 0 | 0 | 0 |
| 16 | 235 | 139 | 76 | 47 | 31 | 22 | 16 | 12 | 10 | 8 | 6 | 5 | 4 | 3 | 3 | 1 | 0 | 0 |
| 17 | 250 | 147 | 80 | 50 | 33 | 24 | 17 | 13 | 10 | 8 | 6 | 5 | 4 | 4 | 3 | 3 | 1 | 0 |
| 18 | 264 | 156 | 85 | 52 | 35 | 25 | 18 | 14 | 11 | 9 | 7 | 6 | 4 | 4 | 3 | 3 | 2 | 1 |

**Figure 2** Projected rediscoveries when DE is known. The number in the $R$ row and $D$ column is the projected number of rediscoveries made in month $R$ caused by DEs discovered in month $D$. The numbers are projected for a hypothetical product that has steady month-by-month growth of usership on the assumption that all users use the initial version of the product.

There are at least two levels on which to approach the decision of whether to do PS. At the coarsest level, one relies on the statistical fact that early-found DEs are the virulent ones, so one does PS for all DEs found up to a certain value of usage, CS for all found later. The considerations as they might be weighed by the developer can be seen by studying a rediscovery matrix, an example of which is shown in Fig. 2. The rows and columns of the matrix are labeled by months reckoned from the time of FCS. The entry for row $R$ and column $D$ is the number of rediscoveries in the user base during month $R$ caused by DEs discovered in month $D$. The total of the numbers in row $R$ is the total number of rediscoveries expected in month $R$.

To construct such a matrix, assume a rate distribution, e.g., the one found from our work. Assign a number of initial latent DEs by the following procedure: Estimate the number of DEs to be found over the usage period of interest for the product of interest, perhaps on the basis of the number of lines of new code in the product; determine a number of initial latent DEs that will give the expected number of DEs for that amount of usage by direct projection from the model. The model is now calibrated to make the matrix.

Use the expected usage buildup for the code to calculate the number of DEs in each rate group that will be discovered in each month after the code is released. To get the column entries for a given month calculate the problem rate of all DEs found in that month as the sum of the number of DEs times the rate per DE. Now, using expected usages for later months, calculate entries for each row of the column as the problem rate of the column month times the number of months' usage for the row month. In the example of Fig. 2 all calculations assume that the initial version of the code is furnished to all users, no matter when they begin to use the code.

The significant trends in the rediscovery matrix are that the numbers grow steadily down a column and diminish strongly to the right across a row. The variation down the column reflects the continual entry of new users who can have problems; the decrease to the right reflects the diminishing virulence of DEs found in later months. It is easy to see that the large numbers all occur in the columns to the left, so that the preponderance of benefits of PS will come from avoiding problems in the leftmost columns.

The solid line in the figure is drawn with reference to a hypothetical PTF tape, made available by the end of month 3 and fixing all DEs found in months 1 and 2. The line encloses all problems whose occurrence can be affected by the fixes on the tape. The hatched area relates to rediscoveries during the period that users are installing fixes, which in this example is assumed to be two months. Some of the problems in the hatched area and all of those below it can be avoided by doing PS to remove the DEs on the tape; no other problems can be so avoided. The value of avoiding these problems must be the justification for doing PS with that tape.

We can construct the line that demarcates the problems affected by some other service vehicle by drawing a similar line with a column determined by the last month for which known DEs will be fixed and the row determined by the month in which the fixes become available to users. As one looks at various cases it becomes clear that, since one needs to eliminate the problems from only a few columns, a vehicle to do the most worthwhile PS can be built only a few months after FCS.

The service strategy for a developer is concerned with deciding what fixes to make available to users with which they can do PS. The optimum strategy for the developer depends on the details of his cost structure, but it is easy to see that he will find it beneficial to support PS to remove only the problems associated with a few columns on the left.

In making the matrix example, we assumed that all new users got the original version of the code unmodified to fix known DEs. The numbers in the rediscovery matrix would be different if, e.g., the developer arranged that each new user get a version of the code in which all available fixes have been installed. In that case one would modify the calculation so that the problem rate for the new users in a given month would be reduced appropriately, and would find that the numbers would not continue to increase down the columns, but would become constant past the month in which all new users have the fixes. For such a service strategy most of the numbers in the matrix will be smaller.

Calculations such as these charts illustrate may be useful in developing quantitative measures for the value of a particular service vehicle in the context of a given planning framework.

A user might approach the PS vs CS decision in a simpler way. He will benefit significantly from PS only if there is a significant chance that a DE he removes preventively would otherwise cause him a problem. Thus he might ask what is the mean time between problems for the DEs that a batch of fixes remove. To determine this one can follow the same calculation as above up to the point of estimating the total problem rate due to DEs found in a given month. If one now divides this quantity by the number of DEs that cause that rate, one gets the problem rate per DE found in a given month. In typical cases the user will find that after not many months the mean time to error for the DEs being fixed by new PTFs approaches the length of time he expects to run the code. The prospective benefit of removing such a DE is very small; and in view of the small chance that the fix will introduce a much more virulent DE, a prudent user will not put in the fix. For typical patterns of usership buildup the average problem rate per DE found in a given month is always about 0.75–0.8 times the reciprocal of the cumulative usage at the end of that month, so that, e.g., DEs found in the month that the cumulative usage reaches 3000 months have an average problem rate of the order 1 per 4000 months.

While the calculations for user and developer look quite different, they lead to much the same cutoff for doing PS, since both calculations are driven by the same underlying consideration: whether the probability of rediscovery justifies the cost of doing PS.

A more refined level of decision about PS can be made if the service organization keeps current records of which DEs caused each problem, since the decision can be made on a DE by DE basis rather than a month to month basis. Assume the developer records the times of occurrence of all problems associated with a given DE and a record of month by month usage of the product. He can then easily make a monthly estimate of the probable virulence of each known DE on the basis of accumulated usage and number of problems caused.

Given this estimate it is straightforward for either developer or user to determine for each DE whether the hazard it poses justifies PS.

One attractive fact about a DE by DE strategy is that application of fixes by all users can be restricted to just the small fraction of DEs for which it is profitable. Since most DEs, even if not fixed by PTF, are never rediscovered, one would like to avoid the cost of doing PS for them. A simple rule ensures this: Wait for at least one rediscovery before doing any PS. Everyone can afford this policy; a highly virulent DE will have its first rediscovery before the fixes could be disseminated anyhow, while a less virulent DE, for which a rediscovery comes only after several months, constitutes a small month by month risk to the users. Another attractive feature of dealing with individual DEs on the basis of estimated virulence is that one deals as efficiently with the (rare) highly virulent secondary DE as with an original DE.

In our judgment the fraction of DEs worth fixing is very small, probably less than ten per cent; and yet for a few DEs the payoff from PS is great. We are convinced that if one determines what PS to do on some such basis as sketched above, one can realize substantially better benefits from PS than are derived from either a fix all or a fix none strategy.

## Concluding remarks

Our work suggests several useful things about the distribution of error rates in product code. First, most of the DEs present have mean times to discovery of hundreds to thousands of months when run on a single machine. Thus the typical DE requires very unusual circumstances to manifest itself, possibly in many cases the coincidence of very unusual circumstances. One may doubt that a testing group using a few machines and having only a few months to work could ever detect and remove all such DEs by purely empirical means. One may conclude that service will always be needed.

Second, if we are correct in our intuition about why the rate distributions in the products we studied were generically similar, we might expect the rate distributions in other large bodies of code to be similar also, at least so long as the methods of code development depend on empirical debugging. It may well be that as software engineering techniques improve, the population of DEs will balance at a lower level; but absent development methods that generate truly error-free code, the same sort of error rate distribution may well persist in future large products.

Third, to the extent that one can assume that the distribution of rates and the total number per thousand lines of code are similar for similar products, one can forecast the error behavior of a planned product. This can be of some use in planning for service. However, one must not expect high precision in such a forecast: There will be large percentage-wise variations in the small number of highly virulent DEs that dominate the early error behavior of the code. These variations will be even larger and less predictable for small products or products that are used sporadically.

Similarly, to the extent that one can represent the error phenomena in the user population by such a simple model, one can use the model for several types of quality control, but again one should do so without expectation of great precision in fitting and interpreting even after-the-fact error data, because the statistics of an actual product history will deviate from most probable values, because the statistical quantities used to interpret the history are subject to large errors of determination, and because the actual history depends on detailed circumstances not represented in the gross parameters. It may well be that our simple model, perhaps as modified to take account of secondary DEs, captures most of the significant regularities in the rate behavior of general pieces of code that can be represented without a lot of detailed information about the use of the product.

## Relation of our work to that of others

The mathematical model and methods we used are simple and familiar. We based certain aspects of our work on the previous work of Richard W. Phillips of IBM Poughkeepsie, but did not make significant use of other work. We did look at some of the literature of reliability, where we found that the general notion of associating problem rates with DEs was a common one. Thus we present this work primarily for what we have learned about the empirical quantification of the error behavior of product quality code and for the implications that can be drawn from it about how to manage preventive service in such code.

## Acknowledgments

In order to do this work we needed the assistance of many people in IBM, who helped us find and get access to the information we needed. We particularly acknowledge the generous assistance of Richard W. Phillips, who by sharing his ideas and the results of his pioneering work on discovery made our task much easier and enabled us to proceed directly along the most profitable path. We also acknowledge that although we present this paper as sole author, all of the work reported was done in collaboration with or depended directly on the results of Gordon Jones, Dan Price, Grant Wood, Alvin Blum, and the other members of their group, whose skill and perseverance in extracting problem data from a jungle of tapes, files, and reports made possible the creation of the problem statistics without which the work could not have been done.

*Received June 10, 1983; revised August 25, 1983*

**Edward N. Adams** *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Adams is presently manager of VLSI symbolic layout tools in the Computer Science Department at the Thomas J. Watson Research Center. Since joining IBM in 1959, he has served twice on the staff of the Director of Research and has served as research director of engineering science, systems and applications, and computer-aided instruction. Prior to joining IBM, Dr. Adams was a faculty member of the University of Chicago's Department of Physics and the Fermi Institute and a member of the Physics Department of the Chicago Midway Laboratories. He was a manager of the Department of Solid State Physics and Semiconductors at the Westinghouse Research Laboratory. Dr. Adams was a visiting faculty member in physics at the Carnegie Institute of Technology, in physics and engineering at the State University of New York at Stony Brook, and in computer science at the California Institute of Technology. Dr. Adams received a B.S. in chemistry from Southwestern, Memphis, Tennessee, in 1943, and an M.S. in physics in 1947 and a Ph.D. in theoretical physics in 1950, both from the University of Wisconsin at Madison. He was an Atomic Energy Commission Fellow from 1948 to 1950. Dr. Adams is a fellow of the American Physical Society and a member of the Association for the Development of Computer-Based Instruction Systems, the Institute of Electrical and Electronics Engineers, and the International Federation for Information Processing.