RC 8228 (#35669) 4/11/80
Computer Science 18 pages

# Research Report

*MINIMIZING COST IMPACT OF SOFTWARE DEFECTS*

E. N. Adams
Computer Sciences Department
IBM Thomas J. Watson Research Center
P.O.Box 218
Yorktown Heights, N.Y. 10598





IBM Research Division
San Jose · Yorktown · Zurich

RC 8228 (#35669) 4/11/80
Computer Science 18 pages

*MINIMIZING COST IMPACT OF SOFTWARE DEFECTS*

E. N. Adams
Computer Sciences Department
IBM Thomas J. Watson Research Center
P.O.Box 218
Yorktown Heights, N.Y. 10598

ABSTRACT

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 benefits a user to the extent that it avoids costly operational problems, but it costs him the resource required to prepare, disseminate, and install fixes; and fixes themselves can introduce errors.

The benefit from removing a given defect depends on how many problems it would otherwise cause. Benefits may be estimated by modelling 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, and that corrective service is preferable to preventive service as a way of dealing with most defects found after code has had some hundreds of usage months of service.

## *SECTION 1. INTRODUCTION*

It is difficult to design and implement a large software system so that it is free of defects (DEs). Accordingly, the producer of the system must normally provide some means to correct DEs after the code has been put into service. As a result where a software system has many users, as does a commercial software product, the users have the opportunity to avoid many problems that might be caused by DEs in the code by installing fixes for these DEs before they cause operational problems.

We shall refer to the correction of a DE in response to a problem as Corrective Service (CS) and the correction before a problem is encountered as Preventive Service (PS). A basic decision that the user must make from time to time is whether CS or PS is the better way to deal with the remaining software defects.

Either approach has its costs. Having to do CS results in the costs associated to operational malfunctions, interruptions, or delays. Doing PS involves costs associated with preparing fixes, preparing and distributing media vehicles for fix distribution, and loss of operational time while installing fixes; and the fixes themselves sometimes inject new DEs in the code.

We shall speak of a service plan as a schedule for the preparation of fix vehicles and of the preventive installation of these fixes in user code. Our interest in this paper is to understand how to formulate a service plan that strikes the best balance between costs and benefits of service at all times in the service life of the code.

Many aspects of service planning can be done in an empirical manner on the basis of operational measurements. In particular, the costs of both CS and PS can be estimated fairly directly on the basis of past experience. In contrast, the benefits of PS, viz., the problems that were avoided, are not directly observable. Thus to estimate the benefits of PS one requires a model of the potential occurrence of problems and of the capability of service to forestall the problems.

We have studied the modelling of service benefits. We shall describe:

> A model of the occurrence of software problems among a user population, their numbers and distribution in time, and how numbers and distribution are affected by PS.
>
> A calibration and evaluation of the model using data obtained from the IBM service organization.
>
> A discussion of how the model may be used to evaluate and optimize a service plan.

## *SECTION 2. MODEL OF PROBLEM OCCURRENCE*

In this section we shall describe a model of the distribution in time of the occurrence of Defect Originating Problems (DOPs) within a population of users of a piece of software.

*basic assumptions and terminology*

defects: A software product contains a number of defects, each of which is contained in some relatively small part of the code. A DE can be removed by installing a fix, which is a corrected version of the piece of code containing the DE.

problem rate: Each DE has some potential to cause user problems. A given DE causes problems at random intervals as though there were a constant fixed probability of occurrence during each unit interval of usage of the code. The average problem rate associated to a DE is characteristic of the DE; in principle it is different for each DE.

problem rate distribution: The distribution of problem rates for one large body of code is similar to that of another developed in a similar manner.

usage: The time dimension in which problems appear is called usage. Usage is related to program execution rather than the passage of time, so usage per unit time will vary with machine size. The unit of usage will be one month of running on a machine of standard size.

multiple releases: A new version of a software product prepared as a modification of an older version with added or changed function will still contain any DEs originally present in the old version and not yet found and fixed.

rediscoveries: During the time that elapses after a DE is discovered and before a fix has been installed in all the user code, the DE may cause further problems for other users. We term these "rediscoveries". The avoidance of rediscoveries is the justification for PS.

*time distribution of discoveries:*

Let U(T) be the rate of usage at time T. Then during the interval dT at T, the probability that a DE having problem rate 1/M will cause a problem is

$$\frac{UdT}{M},$$

and the probability that the DE will survive undetected until the cumulative usage is CU is

$$P(0 \text{ prob} \mid CU) = \exp^{-\frac{CU}{M}}, \tag{2.1}$$

where

$$CU = \int_0^T U(T)dT. \tag{2.2}$$

If the number of original DEs of rate 1/M is ODE , the number of DEs of rate 1/M that will be discovered in time interval dT at T is

$$DR(T)dT = \frac{UdT}{M} ODE \exp^{-\frac{CU}{M}} \tag{2.3}$$

where DR(T) is the discovery rate at time T.

*rate of rediscovery:*

The time rate of rediscoveries of a known DE having rate 1/M is just

$$\frac{U(res)}{M}$$

where U(res) is the residual usage by users in whose code the DE has not yet been fixed.

Calculating the rediscovery rate for a given service plan is a matter of determining the distribution of discoveries in time and the survival of the various DEs in the code as a function of the service delays at the time they are found. This may be done in a straightforward way along lines we shall discuss below. We shall discuss first an idealized service plan, which results in time patterns of problem occurrence that are qualitatively similar to those seen in an actual service situation.

*idealized service plan*

Suppose that when a DE is found the time patterns in which fixes are prepared and then installed are exponential processes having characteristic times L1 and L2. Then, the number DN(T) of DEs that are known but for which fixes are not yet available will satisfy

$$\frac{dDN}{dT} = DR - \frac{DN}{L1}, \tag{2.4}$$

which has solution

$$DN(T) = \int_0^T d\tau DR(\tau) \exp^{\frac{\tau - T}{L1}}, \tag{2.5}$$

and the number DY(T) for which fixes are available but not installed will satisfy

$$\frac{dDY}{dT} = \frac{DN}{L1} - \frac{DY}{L2}, \tag{2.6}$$

which has solution

$$DY(T) = \int_0^T d\tau \frac{DN(\tau)}{L1} \exp^{\frac{\tau - T}{L2}}. \tag{2.7}$$

All DEs that are known, but for which fixes are not installed, are capable of causing rediscoveries, so RR(T), the rediscovery rate at time T, is

$$RR(T) = \frac{[DN(T) + DY(T)]}{M} U(T). \tag{2.8}$$

*numerical example*

Assume that all DEs have a mean time to discovery of 500 usage months, that fixes are got out in 1 month, that they are installed by users in 2 months on average, and that usage grows linearly with time, as is typical of the early months of use of a software product. In this case the usage will be

$$U(T) = gT, \tag{2.9}$$

and the probability that a DE will survive undetected to time T is

$$P(0 \text{ prob} \mid T) = \exp^{-gT^2}, \tag{2.10}$$

where g is defined so that M is in units of 1000 months. The discovery rate is

$$DR(T) = 2gT \text{ ODE } \exp^{-gT^2}. \tag{2.11}$$

In Fig 2.1 we show curves of discovery rate and rediscovery rate vs. time. Note that although the discovery rate is an exponentially decreasing function of cumulative usage, it has a maximum as function of time on account of the linear growth of usage. Note also that the rediscovery rate peaks after the the discovery rate because of the lags in the fix process and the continuing growth of usage.

If in the numerical example we had used a higher problem rate, the discoveries would have peaked earlier and the rediscovery rate would have been higher relative to the discovery rate; if a lower rate, the discoveries would have peaked later, and the rediscovery rate would have been lower.

*distribution of rates*

So far we have been discussing problem occurrence as though all DEs had the same problem rate. However, we shall see that in a real service situation the distribution of problem rates among DEs is quite broad, and defects of a wide range of problem rates are important in an actual service situation.

Although DEs exist because of errors that depend on accidental circumstances of development, we will argue that the statistical distribution of problem rates may be expected to be generally similar among similar products. We theorize: as a developer adds to or changes the code, he continually creates defects, let us say at a rate q; at the same time he continually discovers and fixes defects created earlier. Of the DEs he discovers, some are found while examining the code preparatory to making a change, others while exercizing parts of the code.

Let NDE(M), be the number of DEs having rate 1/M. Then the change of NDE as a result of imbalance between creation and discovery of DEs of rate 1/M is:

$$\frac{d}{dT}NDE(M) = q - \frac{NDE(M)}{T_{insp}} - \frac{\alpha NDE(M)}{M} \tag{2.12}$$

where $1/T_{insp}$ is the rate of discovering an existing DE by examination, $\alpha$ is a coefficient of proportionality, and $\alpha/M$ is the probability of discovering a DE of rate 1/M by exercizing the code. Assuming that NDE will normally reach a steady state value, we may expect the rate distribution on average to approximate the following form:

$$NDE(M) = \frac{qT_{insp}M}{M + \alpha T_{insp}} \tag{2.13}$$

In order to compare Eq. (2.13) to observed rate distributions, one needs to know how the factors q, $\alpha$, and $T_{insp}$ depend on M. We expect $\alpha$ to be independent of M, and we expect $T_{insp}$ to be relatively weakly dependent on M, since we think that the process of discovering a DE by inspection of the code is not closely related to the problem rate associated to the DE. q must vary with M in some way such that the distribution Eqn.(2.13) is normalizable; and we can expect that q cuts off at very small M values, since testing will eliminate a DE whose M is

very small. (In the products we studied this cutoff appeared to come around M = 5-6 months.) We have no other a priori expectations about q; we shall discuss what evidence we could get about the actual variation of q when we compare observed rate distributions to that of Eq.(2.13).

*problem occurrence with a broad rate distribution*

To take account that DEs have a distribution of rates, one treats the DEs in each rate interval as an independent population, solving Eqn (2.1)-(2.8) and determining discoveries and rediscoveries separately for each rate interval. Notationally, ODE will be defined to be the total number of original DEs in the distribution and DF(M) the defect fraction in unit rate range at M. ODE will then represent the characteristic expected to vary widely among products, e.g., due to differences in the numbers of lines of code, while DF will represent the characteristic expected to be similar between comparable bodies of code.

*multiple releases*

A software product typically evolves through a number of versions or levels over a period of years. Although each new release of the product involves new functions and new code, the usual situation is that most of the code of any given version remains present in its successor. We assume that any DEs originally present in a predecessor release will remain in its successor until found and fixed. Accordingly the population of DEs in a given version of a multirelease product will be considered to be the composite of the surviving populations of DEs original in the various predecessor versions, and problem occurrence for the product will be the sum of problem occurrences for the individual populations, the problems for each cohort of DEs being calculated on the basis of the cumulative usage of the code in which that cohort was original.

*effect of service on rediscoveries*

We estimate the effect of planned PS by a computer algorithm that projects all of the rediscoveries that the DEs could possibly cause and then determines for each service vehicle which of the potential problems will be avoided. The algorithm may be visualized as a series of modifications of a rediscovery potential table (RPT). An RPT is shown in Figure 4.1.

To construct an RPT we:

1. Divide the total interval of interest into short periods, for definiteness months.

2. Estimate for each month the number of DEs of each M expected to be found.

3. Calculate for each month the number of rediscoveries expected to be caused by DEs found in the same and each preceding month, and arrange these numbers in a table so that the number in the R,D cell of the RPT is the number of rediscoveries in the Rth month of DEs found in the Dth month.

To take account of the effect of PS on rediscoveries we reduce the numbers in the various cells of the RPT so they reflect only the fraction of users who in the Rth month have not removed the DEs found in the Dth month. We shall illustrate this method below when we discuss applications of the model to service planning.

The treatment of service is straightforward so long as all users do PS the same way. When the user population fragments into subpopulations following different service policies, it may be necessary to separate the RPT into separate parts, and calculate the effects of PS separately for each subpopulation.

*creation of new DEs by PS*

The creation of fixes involves local redesign and rework of the code, so is itself a source of DEs. Fix injected faults are difficult to avoid completely: fixes are typically prepared and disseminated rapidly, so they cannot be tested as thoroughly as original product code, and in some cases fixing a DE has the result that an existing undiscovered DE is unblocked or made more active.

The problem rate associated to a DE created by a fix may have no relation to that associated to the DE fixed, and it is clear from our studies that in many instances fix-created DEs are much more virulent than the DEs they fix. Thus, we must assume that the new DEs created by fixes have themselves a characteristic distribution, perhaps similar to that of the DEs in new products.

In principle to treat the new DEs from fixes one should model the introduction of a fix as the creation of a new version of the code; in practice it would be inconvenient or impractical to do so, since it would involve splintering the code into many versions and the user population into many subpopulations having different numbers of fixes in their code. Fortunately the fraction of fixes that cause problems appears to be small, so that the effects of fix injected DEs can be adequately treated as small corrections which need not be known with high percentage accuracy.

In our approximate treatment of fix injected DEs we follow the course of discovery separately for each subpopulation of DEs having the same rates. We assume that a randomly selected fraction *BF* of fixes create new DEs, and that the problem rates from these new DEs are added to the various subpopulations of DEs over the same time period that the DE is removed by PS. This is an accurate way to model the appearance of new problem activity and at worst somewhat mismodels the effect of PS on rediscoveries of fix injected DEs.

## *SECTION 3. CALIBRATION AND EVALUATION OF THE MODEL*

To evaluate our model of problem occurrence we sought to answer two basic questions:

> How well can the problem rate model reproduce the time patterns of problem occurrence in real systems?
>
> Are the rate distributions required to fit problem occurrence patterns usefully similar from product to product?

To answer these questions we calibrated the model to reproduce discovery and/or problem data for a number of software products. Our sources of data were files of the IBM service organization relating to service experience with these products. We chose to study products for which the data were most extensive, so as to achieve the best possible statistical accuracy

for the rate distributions we determined. For the most part these proved to be system control programs, their components, or similar large bodies of code.

We can summarize our findings thus:

> The problem rate model as we have described it can be made to represent the problem occurrence pattern in any given product fairly well, except that to accurately fit the discovery pattern in the later product life it is necessary to take account of new DEs introduced by fixes.
>
> Rate distributions are sufficiently similar among products of similar type so that an average rate distribution may be used for strategic planning purposes.
>
> In most of our fitting studies the worst variance between fitted and observed discoveries in any time period was typically of the order of 30-50 per cent, and the greatest variations among rate distributions were in respect to the very small number of DEs having very high problem rates. The variances were typically within a factor of 2 of the variances that might be expected from random variations in the processes of creating and discovering DEs. Accordingly we judge that the model, when corrected for fix-injected DEs, captures most of the regularity that is present in the problem data; and we doubt that any simple model using the same level of information would fit the actual data much better.

*data used to calibrate the model*

Our data sources were large files of a sort unfeasible to excerpt here. We can characterize these files as records kept of changes made in master code and records of the effort expended by service engineers in dealing with user software problems.

We extracted three kinds of historical data series to use for model calibration:

1. monthly series of product usage as function of time.

2. monthly series of numbers of DEs discovered.

3. monthly cumulative counts of numbers of DEs found against which 1, 2, ..., DOPs had been reported.

Of these data the second sort could be determined directly from records kept by the repair teams and were highly accurate. The first and third, which were inferred by analysis of service engineer effort reports, are subject to errors of about 10-20 per cent.

One difficulty complicated the analysis: our DE discovery data reflected the problem experience of all IBM service users around the world, whereas our usage and rediscovery data related only to a set of machines in the United States, so pertained to the problem experience of only a subset of the users of the products; also many (sometimes most) of the DEs found in a product were discovered by internal IBM development groups or by users of other products that shared some code components with the product of interest, about whose quantity of usage we had no information.

We dealt with this difficulty by constructing synthetic discovery statistics, in which a DE was counted as discovered only when it had caused a problem within our user population. Our procedure seemed likely to introduce little error in dealing with the high rate DEs, but might distort the numbers of very low rate DEs, which are subject to large error of determination anyhow. We think any errors involved do not cause consequential errors in the rate distributions inferred, but we can test the results only by comparisons of modelled results to synthetic data.

A point of some importance is how best to quantify usage. Intuitively, a quantitative measure of DE-finding usage should be a composite of an interval of time, a number of users, the speeds of the machines used, the duty cycle of use of the code, and perhaps even the type of application being performed. However, not all such information was available.

User numbers we could estimate; and a few cross checks we were able to make, using independently obtained measures of user numbers, showed that our user counts were good.

Dependence of problem rate on machine speed proved difficult to determine, because for every product we studied most machines using it had speeds in the same range. Although we had expected problem rate to be directly proportional to machine speed, we found weak evidence that the problem rate increased less rapidly. We did a regression analysis on the data available. The result suggested a problem rate trend more like the square root of the machine speed, so we assumed a square root relationship in many of our calculations. This assumption does not much affect the fit to observed discovery data, since the spread of machine speeds is not great for any particular product; it does affect the comparison of rate distributions from products typically used by quite different sizes of machine. We are not confident that the problem rate vs. machine speed really has a square root relationship; we think the regression result might in part reflect a statistical difference between the average code quality of two different classes of product.

We had no way to get direct information about the fraction of operational time individual products were used, or about the nature of individual users' applications. Accordingly we simply ignored these factors, and assumed that each of our users gave full time use to the product.

*determining rate distributions*

We used two kinds of analysis to determine rate distributions from usage and problem data. The first of these is based on

$$P(\text{disc} \mid CU,M) = \frac{1}{M} \exp^{-\frac{CU}{M}}, \tag{3.1}$$

where the expression (3.1) is the probability that a DE of rate 1/M will be discovered in unit usage interval when cumulative usage is CU.

Given the discovery times (CU values) for a DE, one may apply Bayes' Theorem and thus invert the probability distribution Eqn (3.1).

$$P(M \mid CU) = \frac{\frac{1}{M} \exp^{-\frac{CU}{M}} P(M \mid 0)}{\sum_{M_i} \frac{1}{M_i} \exp^{-\frac{CU}{M_i}} P(M_i \mid 0)}. \qquad (3.2)$$

Expression (3.2) is the probability that a DE discovered when the cumulative usage is CU will be of rate 1/M.

By adding up the probabilities from Eqn ( 3.2) calculated for each DE that has been found we obtain n(M), the probable number of DE found that have rate 1/M. Finally we can estimate N0(M), the number of DEs originally present that had problem rate 1/M, from

$$N0(M) = \frac{n(M)}{1 - \exp^{-\frac{CU_{tot}}{M}}}, \qquad (3.3)$$

in which $CU_{tot}$ is the cumulative total of usage over the entire interval for which DEs were observed. From N0(M) we calculate ODE, the total number of DEs, and DF(M), the fractions of these having rate 1/M.

A second method of assigning rates to DEs uses the total count of problems for each DE accumulated in some interval beginning with the initial use of the code. The a priori probability that N problems will be caused in interval CU by a DE of rate 1/M is

$$P(N \mid M,CU) = \frac{1}{N!}\left(\frac{CU}{M}\right)^N \exp^{-\frac{CU}{M}}. \qquad (3.4)$$

Inverting by Bayes theorem,

$$P(M \mid N,CU) = \frac{\left(\frac{CU}{M}\right)^N \exp^{-\frac{CU}{M}} P(M \mid 0)}{\sum_{M_i} \left(\frac{CU}{M_i}\right)^N \exp^{-\frac{CU}{M_i}} P(M_i \mid 0)}. \qquad (3.5)$$

Expression (3.5) is the probability that a DE that caused N problems during the usage interval CU is of rate 1/M.

N0(M), ODE, and DF are calculated from Eqn (3.3) as before.

The partitioning of DEs among values of M by the first method is sensitive to the prior values P(M | 0), since the distribution Eqn (3.1) is a very broad one. The partitioning by the second method is sensitive only for those DEs having few or no rediscoveries: DEs for which multiple

rediscoveries were observed can be assigned rather definite rates. Thus, the second method is less ambiguous in determining the very important high rate part of the distribution. When using the second method one must take care that the time interval over which data are accumulated is not one in which appreciable PS is done, else the method will give values that are too low at the crucial high rate end of the distribution; accordingly, when we applied it we used data from only the first few calendar months of code usage.

*rate class approximation*

Lacking a firmly founded parameterized form for the rate distribution, we proceeded as though the shape of the distribution were perfectly arbitrary. Since our data in typical cases were rather meager, we did not wish to use very many adjustable parameters to fit them. We chose to represent the rate distribution as a discrete distribution defined at eight points, so that there were eight parameters to adjust.

We chose the rates each in ratio $\sqrt{10}$ to its predecessor, so the range of variation of rates that could be represented was about 3000 to 1. The highest rate used was such that we could fairly well represent the activity of the most virulent bugs in the data sample; this rate was typically of the order 1 problem per 20 or 30 usage months. The lowest rate used would typically correspond to 1 problem in each 60,000 to 90,000 months of normal usage. This rate is certainly so low as to imply negligible rediscovery, and no doubt we could have cut the distribution off at higher rates with little effect on the rediscovery calculations.

*analysing data from multiple versions of a product*

To determine a rate distribution we required usage and problem data that refer to a single cohort of DEs, introduced together in a single release. However, the usage and discovery data we actually got were most often for a multiversion product, and were categorized by version of code of the user or discoverer, not by code increment in which the DE originated. Thus, the first step in the analysis was to probabilistically apportion the DEs found by users of each version of the code among the various historical increments of code from which they might have come.

We apportioned DEs by a bootstrap method, involving the following steps:

1. Assume a starter set of values of ODE and DF for each historical code increment.

2. Using the starter values calculate the numbers of DEs original in each version expected to be found by the users of each version in each month.

3. Provisionally apportion the actual numbers of DEs found in each month among original versions in the same ratios as were calculated from the starter values. The apportioned DEs now provide provisional discovery histories for each of the individual increments.

4. Fit the provisional discovery histories to determine new values of ODE and DF, using Eqns 3.1 and 3.2.

5. Using the new ODE and DF values, repeat steps 1-4 until the estimates of ODE and DF converge to self consistent values.

The method as described determines a somewhat different function DF for each individual code increment. However, it is basic to the use of this model to be able to use a common original rate distribution for a family of products. Accordingly before iterating as in step 5 above, we determined a set of average DF values for the increments, and used that for all increments in the iterated calculation.

*numerical results*

Using the rate class approximation we determined rate distributions for a number of products. Figure 3.1 presents some sets of DF values found, together with the 8 reciprocal rate values (in units of 1000 Standard Usage Months) that constituted the domain of the distribution. These DF values were all determined using the same principle of assignment of prior probabilities, so are comparable. One can see that the fraction of DEs having very high associated problem rate fluctuates considerably percentagewise from product to product; such quasi-random fluctuations are not surprising, inasmuch as there are very few DEs in the high rate classes. (In some cases less than one DE is assigned to the highest rate class.)

Figure 3.2 shows the set of average DF values for the nine products of Figure 3.1, and, for purpose of comparison, the distribution formulated in Eqn. (2.13), with $\alpha T_{insp}$ taken constant at a value such that the total problem rate of the distribution agrees with that from $DF_{avg}$. The circles shown locate the average of the fitted points (but do not by their size depict the uncertainty of determination of the points).

The points of the average fitted discrete distribution agree sufficiently well with the formulation Eqn. (2.13) that we will for the moment consider that the two have the same analytic form. On that assumption we can determine the dependence of q on M rather simply. We note that by definition q is the number of DEs created per unit of developmental activity per unit interval of M. In our discrete distribution we associate with the single point M all rates having M values between $M\sqrt[4]{10}$ and $M/\sqrt[4]{10}$. This is an interval of length $\gamma M$, where

$$\gamma = \frac{\sqrt{10}-1}{\sqrt[4]{10}}.$$

The observed values are thus to be compared with $\gamma$ M NDE(M).

Assuming that the observed numbers have the same form as the R.H.S. of Eq. (2.13), i.e., are proportional to

$$\frac{M}{M+M_0},$$

we can conclude that

$$NDE_{expt} \propto \frac{1}{M+M_0}. \tag{3.6}$$

But from Eq. (2.13)

$$NDE \propto q\frac{M}{M+M_0}. \tag{3.7}$$

It follows that to the approximation of our analysis

$$q \propto \frac{1}{M}. \qquad (3.8)$$

Eqn.(3.8) decribes a variation of the generation of DEs during development that is the geometric mean of one that is constant per unit interval of 1/M and one that is constant per unit interval of M.

Despite the noisiness of the data, we have some confidence that our observations capture a real trend in the data, because we experimented with many combinations of prior conditions and processing algorithms and found similar trends of the results in the important part of the distribution. However, it is difficult to estimate the probable error of determination, in view of the fact that the determination of the distribution at large M is significantly affected by the choice of prior probabilities.

To indicate the degree of fit of the model we shall present data for a typical example. Figure 3.3 shows DF values for three successive versions of a software product as determined from an analysis of the discovery data, and Fig 3.4 compares fitted and observed time patterns of discovery. We think the data for the first month or so of use of version 1 are incomplete, and that causes the calculated rate distribution for that version to be somewhat impoverished of high rate DEs. Even so, the DEs projected for the three versions using a single average DF provide a reasonably good representation of the discovery history.

*determining actual service lags*

We had accurate information about when fixes were made available, but we had no direct information about when or how thoroughly users performed PS. While such information was not essential to calibrate the basic model, we were interested to determine how well the model represented the effects of PS. We found it possible in a few cases to estimate, in a manner we shall describe, roughly how much PS had been done.

Evidence that PS was being done can be seen from the trends of the data of Figure 3.5, which presents sets of DF values determined for a single software product after each of the early months of its use, using the method of problem count analysis. The distribution may be seen to be fairly stable for the first five months, after which the fraction of high rate DEs steadily diminishs. The reason for the trend seen is clear: the data analysis is made on the assumption that no PS is done. So long as that is true, the month by month increase of reported problems caused by the highest rate DEs goes up, more or less in proportion to the actual usage. After awhile most of the users have installed fixes for the early found DEs, so few or no additional problems are reported against them anymore; now as time goes on the allocation algorithm assigns these DEs to lower and lower rate classes, because the same number of problems is associated to larger and larger amounts of cumulative usage.

One can quantify the effects of PS by doing analysis on a DE by DE basis: one can follow each DE from its time of first discovery, note the maximum apparent rate associated to it and the usage value at which that occurs, and on the basis of this information make a probabilistic assignment; doing so, one can get a good picture of the high rate part of the original rate distribution, and can also observe how and when PS becomes effective by determining how the problem activity for each DE decays after it passes its peak.

We did not actually do much of this kind of analysis because, it is only semiquantative: we were not able to determine service parameters uniquely in a given case, and we were told by service engineers that there was no single typical service pattern about which to generalize

from product to product. (Our data tends to confirm this.) We did fit rediscoveries for a few products, and found that we could fairly well represent the trend of rediscovery data, if we made suitable assumptions about user practice and also took account of the new DEs injected by fixes. When rediscoveries have been apportioned among original code increments, their time variations in general resemble the rediscovery curve depicted in Fig.(2.1).

*discussion*

There are important limitations on the accuracy with which a complete rate distribution can be determined from observational data, especially with respect to the low rate part of the distribution. Absent a parameterized theoretical form of the distribution, so that measurements of one part of the distribution give information about other parts of it, one can only know the rate associated with a DE by observing it until it has been rediscovered a number of times. For the products we studied, many or most DEs caused only a single problem over all the time the code was in use, either because their associated rates were very small or because PS removed them before they could be rediscovered. The probability distribution for assigning rates to such DEs is very broad, so the assigned rates are quite sensitive to the prior probabilities assumed.

Forecasts for a new product are subject to large error in regard to the initial problem rates because this rate gets a large contribution from the small number of high rate DEs present, and the number of these are subject to large percentage fluctuation. Even if the model were perfectly correct and if a perfectly accurate rate distribution were known, one could expect to fit the model to real world service events with only medium accuracy in view of the magnitude of expected random fluctuations in the problem occurrence process. For the products we studied the model ordinarily fit the data with an accuracy no more than a factor 2 worse than would be expected for a perfect model.

## *SECTION 4. EVALUATING A SERVICE PLAN*

*general considerations*

The basic goal in service planning is to find the most profitable combination of CS and PS. Two principles summarize what is important about the process:

> The greatest part of the problem rate is associated to a relatively small number of high rate DEs, which are typically found very early in the life of the code. PS is profitable chiefly to the extent that it removes such DEs.
>
> For dealing with a DE having a problem rate lower than some critical problem rate, it is less costly to use CS than PS; this will be true for the average DE found when the code is old.

The reason why it becomes unprofitable to do PS when the code has been stabilized beyond a certain point is that as the code gets older the mean problem rate removed per DE fixed continues to diminish, whereas the cost per DE of doing PS does not; and, indeed, if PS were continued indefinitely, eventually the fixes would inject more problem rate than they removed.

A service plan for a complex service situation will specify the number of code versions to be released, their time of release, and for each, the number of DEs to be initially present, the distribution of rates among them, the amounts of usage the code will be subjected to in each time interval of interest, and the expected schedule of PS. If every service vehicle applies to all extant versions of the code, one evaluates the benefits of PS using a single RPT that is a composite of the RPTs for DEs of the individual code increments; otherwise, one uses a separate RPT for each code increment.

As discussed above we determine the effect of the planned service on the rediscoveries by means of a computer algorithm that reduces the RPT entries according to the planned schedule of PS. When all reductions due to PS have been made, the modified RPT is compared to the original to determine the benefits realized. The benefits so calculated will depend on the assumed usership profile over the entire life of the code. We shall discuss this further by means of examples.

*value of a single service vehicle*

Figure 4.1 shows a RPT computed for a hypothetical example that might be typical of a large product in a many user system. In this example a single version product is serviced by a single service vehicle released at the beginning of the fourth month, which provides fixes for all DEs found in the first two months, and the users install these fixes over a period of two months after they become available.

The solid line in the table to the right of the second column (the last month fixed) and above the fourth row (the first month the fixes are available) demarcates the sets of potential rediscoveries that could conceivably be affected by this service vehicle. All rediscoveries within this region could in principle be eliminated if users installed fixes as soon as they were available. As it is, the rediscoveries in the hatched region ( first two months after fix is available) are only partly avoided.

One general service practice that has much to recommend it is to do all desired PS on the master code issued to new users as soon as fixes become available, without waiting for release of service vehicles. This practice is most valuable during the early months of usage, when usage is building up at a maximal rate and when the highly virulent DEs, whose removal provide the chief justification for PS, are being found. Normally this practice will have a low marginal cost to implement and will yield substantial benefits. In the examples below we shall calculate the RPT on the assumption that the master code is routinely cleaned up in this manner. The magnitude of the benefit from doing so can be seen by comparing the RPTs of Figure 4.1 and 4.2. Note that in Figure 4.2 the numbers down a column reach a maximum value and remain there; realistically these numbers should slowly decline from the maximun as the DEs are removed by CS.

*optimizing a service plan*

Optimizing a service plan is in principle a matter of specifying all candidate plans, calculating their costs and benefits, and choosing the best one. In practice one is likely to begin from some base plan and study what variations might make it more profitable. Some features of a plan are likely to be viewed as necessary constraints, not subject to modification. Thus, perhaps because of operational rhythms or external commitments, one might be committed to particular service vehicles, but be in doubt when or whether to issue others. In such cases one

may wish to know the marginal benefits that would be contributed by individual service vehicles which one is not firmly committed to bring out.

If, as in the example above, there were only a single service vehicle, we would naturally assign all of the benefits of PS to it. Where more than one service vehicle is used, the assignment of benefits to particular vehicles is ambiguous, since the parts of the RPT affected by different service vehicles overlap. This is evident from Figure 4.2, which is based on the same product scenario as Figure 4.1, except that master code is continually updated.

Figure 4.2 is prepared as though one were considering a new service release every month during the early life of the code. Suppose that one were uncertain whether it would be worthwhile to issue the first release. One could determine from Fig. 4.2a that certain of the problems potentially avoided by PS, viz., 11 problems in the 3,1 cell, 22 problems in the 4,1 cell, and 11 problems in the 5,1 cell, would be avoided only if the first vehicle were released: the others would be avoided if the other service vehicles were released, whether or not the first vehicle was released. In this context, then, these 44 problems are the only benefits that can be unambiguously ascribed to the first service release; so one might reasonably decide whether to bring out the first release by comparing the cost of CS for these 44 problems with the cost of bringing out the release and doing PS with it.

In order conveniently to examine such questions, we have used an algorithm that calculates the benefit from the modified RPT after the effect of each service vehicle is determined, so that we learn not only the total benefit derived from the plan, but the incremental benefits from the individual service vehicles taken in order. The benefits assigned in this way to a given vehicle depend on its position in the order of evaluation; so we arrange the vehicles in order of priority according to the degree of commitment that exists to release them.

The priority assigned to a given vehicle depends on what has to be decided. In the example above, where the question was whether to issue the earliest possible first service vehicle, that vehicle would be assigned last priority, because it was the marginal action one considered omitting. However, suppose that the decision had been made to bring out the first service release as planned, and the question next became whether the second monthly release should be brought out. In that case the first release would be assigned first priority and the second release last priority.

The above examples should indicate the general procedure for calculating the benefits from planned PS.

## *CONCLUDING DISCUSSION*

Our principal findings with regard to problem occurrence in software products:

1. The observed pattern of rediscoveries of DEs is consistent with the simple concept that each DE has an intrinsic problem rate per unit of usage.

2. The distributions of problem rates are similar for a number of large software products, and takes a form that suggest that the number of DEs in the code results from a statistical balance during development between creation and discovery by execution of the code.

The form of problem rate distribution we have found has general implications with regard to best user strategy in doing PS. Typically we found that the 10 per cent of DEs having the highest problem rates account for more than 90 per cent of all rediscoveries, and are almost all found during the first few months of use of the code. As a result most of the DEs that are profitable to remove by means of PS are known very soon after the code is put into use, and can be removed by a few promptly prepared and installed early releases. Accodingly, after a few months or quarters, depending on the size of the user population, a user may find it better to discontinue across-the-board application of preventive fixes for all DEs.

In even old code an occasional DE will cause significant numbers of rediscoveries. Such DEs are probably all injected or activated by fixes. Even though the number of fixes that introduce new DEs is small and the typical DE introduced is not very virulent, when a fix does create a DE that is highly virulent, one may wish to remove it by PS. However, one does best to limit PS on old code to DEs that are highly virulent; so one requires a criterion that a DE is virulent enough to need fixing.

A criterion for removing a DE may be formulated in terms of the number of future rediscoveries expected if no PS is done:
One estimates the problem rate associated to the DE by determining its rediscovery rate immediately after discovery; then one estimates the expected number of rediscoveries over the expected usage life of the code if the DE isn't fixed; finally one estimates whether costwise it is better to deal with these expected rediscoveries by CS or avoid them by PS. A practical criterion will usually be to preventively fix every DE that is rediscovered one or a few times within a few months.

## *RELATION TO THE RESULTS OF OTHER WORKERS*

Both our model and our methods of analysis are familiar ones. Our exponential model for discovery rates is that of the radioactivity of a material containing a mixture of radioactive species, the discovery of a DE being the analog of the decay of a radioactive atom. The use of execution time rather than chronological time is fundamental, since the objective is to forecast the rediscoveries in the user population. The use of discovery rates that are exponential in execution time is a familiar one in software reliability theory, where it is sometimes referred to as a Musa[1] model.

Our objective was to develop a forecast model, i.e., a model that would project future problems for a product before it is in use. Several software reliability forecast theories recently been reviewed by Swearingen and Donahoo[2]. These theories typically seek to predict the pattern of errors from the size and complexity of the code of the product and/or characteristics of the program development technology. Our basis of prediction is of a somewhat different sort. Because our model is intended for use in the planning phase of the product, it presupposes only general knowledge about the product, e.g., the approximate amount of new code and perhaps some information about the code of a predecessor product; thus, its inputs are the number of DEs present when the code is put into use (estimated from the amount of

new code) and the distribution of problem rates (estimated from measurements on similar products).

In its focus on rediscoveries and the effect of service to avoid them our work is different from other work we are aware of. We think that in drawing implications about the relative value of early and late PS we are working in an area that hasn't been systematically treated in the literature. ( However, see Littlewood[3], which clearly points out the way in which the distribution of rates will be important.)

In the actual carrying out of our work we found the unpublished work of Phillips[4] to be of the greatest value. The case for doing our work in the first place was strengthened by Phillips' finding that the error discovery histories for several products were in approximate agreement with a universal usage vs. per cent discovery relation; and from him we got our first information about the magnitude of error rates in actual product code and about possible data sources and methods of data analysis.

## *ACKNOWLEDGEMENTS*

In order to do this work we needed the assistance of many IBM people, who helped us find and get access to needed information. We wish particularly to express our thanks for the generous assistance of Dick Phillips, who by sharing with us his ideas and the results of his pioneering work on DE discovery made our task much easier than it would otherwise have been; and we wish to acknowledge that, although we present this paper as sole author, all of the work reported here either was done in collaboration with or depended on the results of Gordon Jones, Dan Price, Grant Wood, and the other members of their group, whose perseverance and eventual success in extracting valid problem information from a jungle of files and reports alone made it possible to determine problem rates in the software products we studied.

## *REFERENCES*

1. J. D. Musa, "A theory of Software Reliability and its Applications", IEEE Transactions on Software Engineering, Vol. SE-1, No. 3, pp. 312-327.

2. D. Swearingen and J. Donahoo, p. 151, Proceedings of an IEEE Workshop on Quantitative Software Models, Oct 9-11, 1979, IEEE Catalog No. TH0067-9, Library of Congress Catalog No. 79-91318.

3. B. Littlewood, p. 170, ibid.

4. R. W. Phillips, private communication.

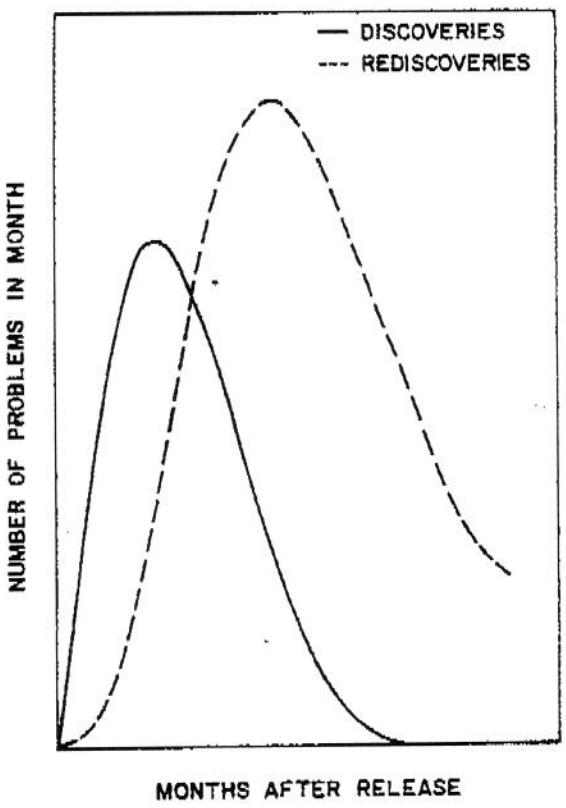


Figure 2.1. Typical pattern of discovery and rediscovery rates

| | 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 | .6 | .19 | .06 | .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 |

Figure 3.1. DF values for a number of software products

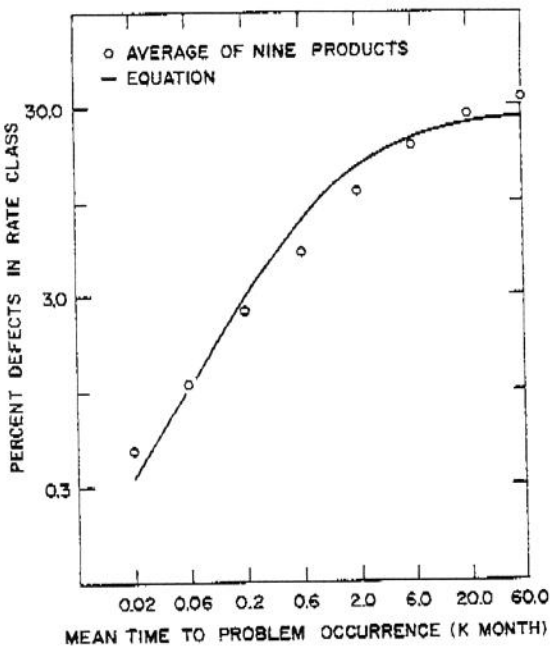


Figure 3.2. Average observed DF vs. distribution of Eqn. 2.14.

| | RATE CLASS | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| MEAN TIME TO PROBLEM OCCURRENCE IN KMONTHS FOR RATE CLASS | 95 | 30 | 9 | 3 | .9 | .3 | .09 | .03 |
| FITTED PERCENTAGE DEFECTS IN RATE CLASS BY RELEASE | | | | | | | | |
| REL | | | | | | | | |
| 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 |
| AVG | 22.4 | 24.6 | 27.3 | 14.5 | 4.0 | 4.8 | 2.2 | 0.3 |

Figure 3.3. DF for three successive versions of a product.

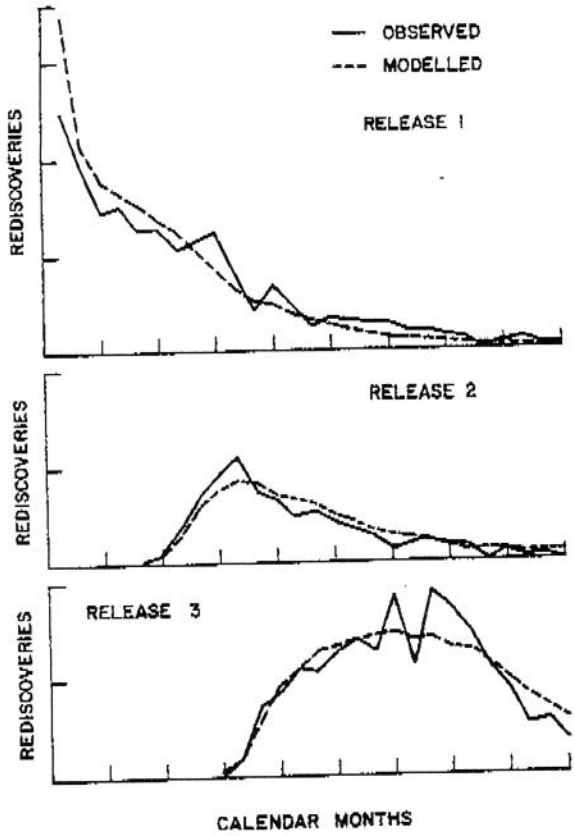


Figure 3.4. DEs found vs. time for 3 versions of a product.

Observed and fitted discoveries vs. time are shown for the three versions of the product of Figure 3.3.

PERCENTAGE OF DEFECTS BY RATE CLASS

| CLASS / MONTH | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 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 |

Figure 3.5. Variation of empirical DF with time.

Because of PS, calculated DF values determined at different times after the product was put into use show varying amounts of very high rate activity.

| REDISCOVERY MONTH | DISCOVERY 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 4.1. Rediscovery Potential Table for a simple example.

A single service vehicle is made available at the beginning of the 4th month fixing all DEs found in the 1st two months. The solid line demarcates the problem events potentially affected by this vehicle. The hatched area shows the problems that are only partly avoided if the users take 2 months to get the fixes installed.

| REDISCOVERY MONTH | DISCOVERY MONTH 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 26 | 7 | 0 | 0 | 0 | 0 | 0 |
| 4 | 44 | 35 | 19 | 6 | 0 | 0 | 0 | 0 |
| 5 | 44 | 35 | 24 | 15 | 5 | 0 | 0 | 0 |
| 6 | 44 | 35 | 24 | 17 | 12 | 4 | 0 | 0 |
| 7 | 44 | 35 | 24 | 17 | 14 | 10 | 4 | 0 |
| 8 | 44 | 35 | 24 | 17 | 14 | 11 | 8 | 3 |
| 9 | 44 | 35 | 24 | 17 | 14 | 11 | 9 | 7 |
| 10 | 44 | 35 | 24 | 17 | 14 | 11 | 9 | 8 |
| 11 | 44 | 35 | 24 | 17 | 14 | 11 | 9 | 8 |
| 12 | 44 | 35 | 24 | 17 | 14 | 11 | 9 | 8 |

| REDISCOVERY MONTH | DISCOVERY MONTH 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 9 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 33 | 26 | 7 | 0 | 0 | 0 | 0 | 0 |
| 4 | 11 | 26 | 19 | 6 | 0 | 0 | 0 | 0 |
| 5 | 0 | 9 | 18 | 15 | 5 | 0 | 0 | 0 |
| 6 | 0 | 0 | 6 | 13 | 12 | 4 | 0 | 0 |
| 7 | 0 | 0 | 0 | 4 | 10 | 10 | 4 | 0 |
| 8 | 0 | 0 | 0 | 0 | 5 | 8 | 8 | 3 |
| 9 | 0 | 0 | 0 | 0 | 0 | 4 | 7 | 7 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 6 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figures 4.2a and 4.2b. RPTs for monthly service releases.

Fixes are made available 1 month after DEs are found, and installed by users over a period of 2 months. The various lines in 4.2a demarcate the events potentially affected by the various service vehicles. 4.2b shows the RPT modified to take account of all PS.