Predicting Fault Incidence Using Software Change History

Todd L. Graves, Alan F. Karr, J.S. Marron, and Harvey Siy

Abstract—This paper is an attempt to understand the processes by which software ages. We define code to be aged or decayed if its structure makes it unnecessarily difficult to understand or change and we measure the extent of decay by counting the number of faults in code in a period of time. Using change management data from a very large, long-lived software system, we explore the extent to which measurements from the change history are successful in predicting the distribution over modules of these incidences of faults. In general, process measures based on the change history are more useful in predicting fault rates than product metrics of the code: For instance, the number of times code has been changed is a better indication of how many faults it will contain than is its length. We also compare the fault rates of code of various ages, finding that if a module is, on the average, a year older than an otherwise similar module, the older module will have roughly a third fewer faults. Our most successful model measures the fault potential of a module as the sum of contributions from all of the times the module has been changed, with large, recent changes receiving the most weight.

Index Terms—Fault potential, code decay, change management data, metrics, statistical analysis, generalized linear models.

◆

1 INTRODUCTION

As large software systems are developed over a period of several years, their structure tends to degrade and it becomes more difficult to understand and change them. Difficult changes are excessively costly or require an excessively long interval to complete. In this paper, we concentrate on a third manifestation of "code decay": when changes are difficult in the sense that excessive numbers of faults are introduced when the code is changed. As the system grows in size and complexity, it may reach a point such that any additional change to the system causes, on the average, one further fault, at which point, the system has become unstable or unmanageable [1]. This paper is devoted to identifying those aspects of the code and its change history that are most closely related to the numbers of faults that appear in modules of code. (In this paper, the term "module" is used to refer to a collection of related files.) Our most successful model computes the fault potential of a module by summing contributions from the changes ("deltas") to the module, where large and/or recent deltas contribute the most to fault potential.

We use only information available on 31 March 1994 in our models and these models predict number of faults that appeared between 1 April 1994 and 31 March 1996. We find that the change history contains more useful information than we could have obtained from product measurements of a snapshot of the code. For example, numbers of lines of code in modules are not helpful in predicting numbers of future faults once one has taken into account numbers of times modules have been changed. This implies that many software complexity metrics are also not useful in this context because, within our data set, these metrics are very highly correlated with lines of code.

A measure of the average age of the lines in a module can also help predict numbers of future faults: In our data, roughly two-thirds as many faults will have been found in a module which is a year older than an otherwise similar younger module.

In addition to size, other variables that do not improve predictions are the number of different developers who have worked on a module and a measure of the extent to which a module is connected to other modules.

After discussing the variables available in our data and their peculiarities in Section 2 and, then, describing some of the statistical tools that we will be using in Section 3, we present our models of fault potential—that is, the distribution of faults over modules—using data available at the beginning of the prediction interval in Section 4.

1.1 Software Fault Analysis

Previous work in software fault modeling can be classified into prediction of number of faults remaining in the system and accounting for the number of faults found in the system.

Most of the work on prediction is done in connection with software reliability studies in which one first estimates the number of remaining faults in a software system, then uses this estimate as a predictor of the number of faults in some given time interval. (See Musa et al. [2] for an introduction to software reliability.) Classic models of software faults [3], [4] are discussed in a survey of the early work on measuring software quality by Mohanty [5] which has a section on estimation of fault content. There are

- *T.L. Graves is with the Los Alamos National Laboratory, MS F600, Los Alamos, NM 87545. E-mail: tgraves@lanl.gov.*
- *H. Siy is with Lucent Technologies, Naperville, IL 60566. E-mail: hpsiy@lucent.com*
- *A.F. Karr is with the National Institute of Statistical Sciences, PO Box 14006, Research Triangle Park, NC 27709-4006. E-mail: karr@niss.org.*
- *J.S. Marron is with the University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3260. E-mail: marron@stat.unc.edu.*

also many recently proposed models [6], [7]. These models estimate the number of faults that are already in the software. Our work differs from these studies in that we assume new faults are continuously being added to the system as changes are made.

Our research is similar to previous empirical studies of software faults, e.g., [8], [9], [10], [11], where the aim is to understand the nature and causes of faults. In these studies, explanatory variables are identified in order to account for the number of faults found. Our work extends this by attempting to understand how the process of software evolution could have an effect on the number of faults found over time. In several of the cited studies, no actual model is articulated. Our goal is to build fault models based on these explanatory variables that are reasonably accurate and interpretable.

Below, we list some of the factors, cited in previous work, which were thought to be predictors of number of faults. These factors can be classified into two groups: product-related and process-related measures. Within each group, we will list the measurements in that group that we used to try to predict fault potential and describe how successful these measurements were.

1.2 Product Measures

Product measures can be computed using syntactic data taken from a snapshot of the software. These include, for example, code size (lines of code) and degree of statement nesting. Several studies have shown that large modules have lower defect densities than small modules [8], [9]. Hatton [10] reports that the decrease in defect densities is not linear, but is U-shaped, implying that there are medium-sized components that have lower defect densities than large components, which in turn have lower defect densities than small components. Other studies have found that modules with a high amount of nesting also tend to have more defects [8].

Other product measures are measures of code complexity, like McCabe's cyclomatic complexity [12] and Halstead's program volume [13]. Schneidewind and Hoffman [14] compared several measures of complexity, among them cyclomatic complexity, number of acyclic execution paths, and number of ways to reach a program block, and found that, regardless of the complexity measure, programs with high complexity have a high number of faults and, similarly, programs with low complexity have low number of faults. In a study to identify fault-prone modules in a telecommunication system, Ohlsson and Alberg [15] found that modules with high cyclomatic complexity are likely to have more faults. Shen et al. [16] and Munson and Khoshgoftaar [17] found that Halstead's η_1 (number of operators) and η_2 (number of operands) are the best indicators among Halstead's other metrics and better than cyclomatic complexity.

The product measures we study in this paper include:

- *lines of code* (both commentary and noncommentary lines) at the start of the prediction interval,
- *other complexity metrics*, computed using an in-house complexity metric tool.

Our own investigation into these measures showed that each one was highly correlated to lines of code which (if change history is available) is not a very good predictor of faults.

1.3 Process Measures

Process measures are computed using data taken from the change and defect history of the program. The simplest measure classifies modules as new or modified code: Basili and Perricone [9] found, for example, that new and modified modules behaved similarly except for the types of faults found in each and the effort required to correct errors.

Another measure cited in literature is the number of defects already found in each module. Yu et al. [11] found that modules with histories of large numbers of defects are likely to continue to be faulty.

Some other related work has concentrated on predicting the numbers of faults that remain in software part way through a corrective maintenance phase. One such technique measures the amount of overlap in defects found by different people during a code inspection [6]: The higher the overlap, the less likely a module is to have more defects. Christenson and Huang [7] performed a study to predict the number of remaining faults by counting "fix-on-fixes" (fault fixes that become faults themselves): The fewer faulty bug fixes detected, the less likely a module is to have more defects.

We use several additional novel measures derived from the change history of the software to predict the number of faults. Descriptions of these measures follow.

- Number of *past faults*. Our "stable" model predicts the number of faults to be found in a module in the future to be a constant multiple of the number of faults found in a period of time in the module. We used this model as a yardstick against which to compare other models and we found that it was challenging to improve upon it. A potential reason for a module to have contained a large number of faults in the past is that it was tested more rigorously than other modules, in which case it might be expected that it would be relatively fault-free in the future. Our data did not contain information about testing effort across modules; in any case, the success of the stable model would tend to indicate that different testing intensities were not an important factor.
- Number of *deltas* to a module over its entire history. The number of changes to code in the past was a successful predictor of faults, clearly better than product measures such as lines of code. A module's expected number of faults is proportional to the number of times it has been changed.
- A measure of the average *age* of the code, calculated by taking a weighted average of the dates of the changes to the module, weighted by the size of the changes. This measure, when combined with number of deltas, greatly improved the fit of the model, to the point where it is as good as the stable model and, possibly, slightly better.

- The development *organization* that worked on the code. The code we study was developed by two organizations that define numbers of faults in different ways, and this required us to put a nuisance parameter in the model. See Section 2.1 for details.

- The *number of developers* who have made deltas on the module. Perhaps surprisingly, there was no evidence that a large number of developers working on a module caused it to be more faulty.

- The extent to which the module is *connected* to other modules, as measured by the typical number of other modules changed together with the module, also did not appear to be important in our models, although we expected that large numbers of interfaces would be characteristic of code that is difficult to change correctly.

- A *weighted time damp* model, which computes a module's fault potential by adding contributions from each change to it, with a change contributing a lot of fault potential if it is large and recent. This was the most successful model we located. This model allowed us to conclude that the rate at which changes' contributions to fault potential disappear with time is about 50 percent a year.

2 CHANGE MANAGEMENT DATA

In this paper, we study the code from a 1.5 million line subsystem of a telephone switching system. We predict the incidence of faults in each of 80 modules in the subsystem. A few hundred developers changed the code a total of roughly 130,000 times. We predict numbers of faults in a two-year period from various data.

The data came from two sources: an Initial Modification Request (IMR) database and a delta database. (An IMR is the official record of a problem to be solved. Solving an IMR will typically lead to several modification requests, or MRs, which are assigned to specific developers. MRs typically consist of several deltas, each of which is an editing change to a single file.)

2.1 IMR Database

The IMR database lists, for each IMR, the date that the request was first made ("open date"), the date that the last delta associated with that IMR was completed, the person who originated the IMR, whether the IMR was classified by the originator as "bug" or "new" (a bug fix or a new feature, respectively), and a list of the modules changed as a result of the IMR.

To understand the IMR data, it is helpful to know the development process followed by the software organization. The software organization is composed of two nearly autonomous organizations, one maintaining a domestic (U.S.) version of the product and the other maintaining an international version. For the most part, the processes followed by the two are similar. (There are certain differences that we will point out.)

Features are the fundamental unit of development. In other words, developers are always doing development work, going from one feature to another. Requests for new features are sent to the software organization as sets of IMRs classified as "new" IMRs. Working on an IMR involves the usual engineering activities of requirements specification, design, and coding. The artifacts of requirements, design, and coding are subjected to formal reviews and inspections. After that, the features are unit tested and then submitted to the integration team, which builds the various features into a software release. The features are then subjected to integration and regression testing. Problems uncovered in the reviews and inspections are fixed as part of the work for the original IMR. The exception is, if the issue raised is not directly relevant or required for this feature, it is documented as future work through another "new" IMR. This procedure also holds for problems found in testing and integration within the organization maintaining the domestic product. For international development, problems uncovered during the integration and testing stages are documented as "bug" IMRs and sent back to the developer who wrote the code. Finally, problems found in the field are reported back to both development organizations as "bug" IMRs. The difference between the two organizations' bug reporting procedures leads to a large discrepancy in the fault rates of modules developed by the two organizations because the international organization would report an average of four faults for a problem that would only prompt one fault report for the domestic organization.

We restricted our study to the IMRs classified as bug fixes and defined our response variable to be the number of bug IMRs which touched a given module and whose open dates were in the period 1 April 1994 through 31 March 1996. During this period, there were more than 1,500 IMRs that were classified as fault fixes. Of these, 15 percent were faults reported in the field, while the rest came from various releases still under development. (At any given time, the development organization manages several active releases. Some of these releases are still in development, while others are in the field.) Fig. 1 contains a plot of the density of fault IMRs in time for a single module (the module which had the most faults in the prediction interval).

A potential weakness of this choice of response variable is that IMRs classified as bugs may not always reflect faulty software: For instance, if code is changed after it is released because it failed to include some desired feature, the changes are classified as bugs even if the code performs perfectly outside the missing feature.

The IMR data set was also used to construct the "stable model" discussed in Section 4 with which we predict future fault IMRs using past fault IMRs; this model serves as a point of reference for other models.

2.2 Delta Database

The source code for this system is under change control using SCCS [18]. There is a change management database to keep track of the set of deltas associated with each IMR. This change management database records several attributes of the deltas, including the date (to the second) of the change; the names of the module and file within the module that are changed (each delta can affect only one file); the numbers of lines added, deleted, and left unchanged in the file by the delta; the identity of the developer making the
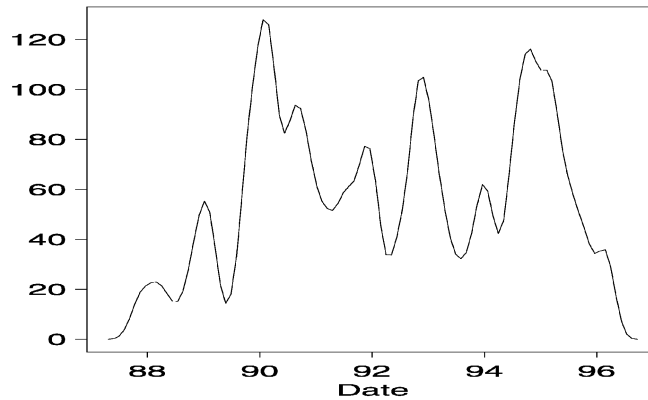
Fig. 1. Density of faults per year for the most faulty module.

change; and an identifier to link to the IMR. This change management database is the basis of our second data set from which we drew most of the measurements used as independent variables. From this data set, we computed such quantities as the length in lines of code of the modules at the beginning of the prediction interval.

2.3 System Characteristics

The system we studied is a legacy system, with at least one version released every year. Much of the code is written in C, but about half the modules contain files written in a domain-specific language. Several different versions of the code can exist at the same time because MRs may be applied to some versions, but, at least not immediately, to others. For example, the subsystem we studied is released in dramatically different versions to domestic and international customers: of the 80 modules, roughly 20 are included in the international package alone, 30 are included in only the domestic package, and 30 "common" modules are included in both. The domestic code (for example) is released in a different form every year. We did not explicitly include these considerations in our models except to allow fault potentials of modules to differ depending on whether they are part of the domestic or international packages or both.

2.4 The Modules

Of the subsystem's roughly 100 modules (themselves collections of files), roughly 20 were removed from our analyses for various reasons (for example, if they contained only header and make files). Two modules not yet in existence at the beginning of the prediction interval were also removed; predicting faults in these modules could be done using, for example, [15], which employs software complexity metrics calculated from design documents.

The remaining 80 modules had a total of about 2,500 files (between three and 155 apiece). The modules contained about 1.5 million lines at the beginning of the prediction interval, with the smallest module containing about 1,000 lines and the largest about 100,000. Before the prediction interval, the 80 modules were touched by between 30 and 7,000 deltas apiece. Eighteen of the modules were fault-free in the prediction interval, while four modules were affected by more than a hundred fault IMRs. The total number of fault IMRs is more than 1,500. Fig. 2

contains histograms of the numbers of faults in the modules in the prediction interval, the lengths in lines of the modules at the start of the prediction interval, and the numbers of deltas to the module before the start of the prediction interval.

3 STATISTICAL TOOLS

In this section, we present some of the statistical techniques we used to perform these analyses. The modeling was done using generalized linear models as described in Section 3.1. Our choice of parametric family to use led to some complications and forced us to use simulation to assess the amount of uncertainty in our estimates; see Section 3.2.

3.1 Generalized Linear Models

Generalized linear models (GLMs) [19] extend the ideas of linear regression. Whereas linear regression assumes that the expected value of a response variable is a linear function of one or more predictors and works best if the error distribution is nearly normal and if the variances of the observations do not depend on the means, in GLMs, a function (the "link" function) of the mean is linear in the predictors and other error distributions, such as Poisson, are allowed.

In our analyses, we used a logarithmic link and took the error distribution to be Poisson. (In this paper, $\log$s are always base e.) This means that if we predict number of faults using the log of the number of lines of code in the module, then the number of faults has a Poisson distribution with mean equal to a constant multiple of some power of the number of lines.

One way of looking at what a generalized linear model is doing is by considering the error measure it uses. Since the numbers of faults in the modules differ by orders of magnitude, one should take care in choosing an error measure for evaluating the quality of a model. It is critical that neither large modules nor small modules have an inordinate effect on our evaluation of a model. We found that, for a model predicting numbers of faults $y_1, \ldots, y_n$ using predictors $e_1, \ldots, e_n$, the error measure

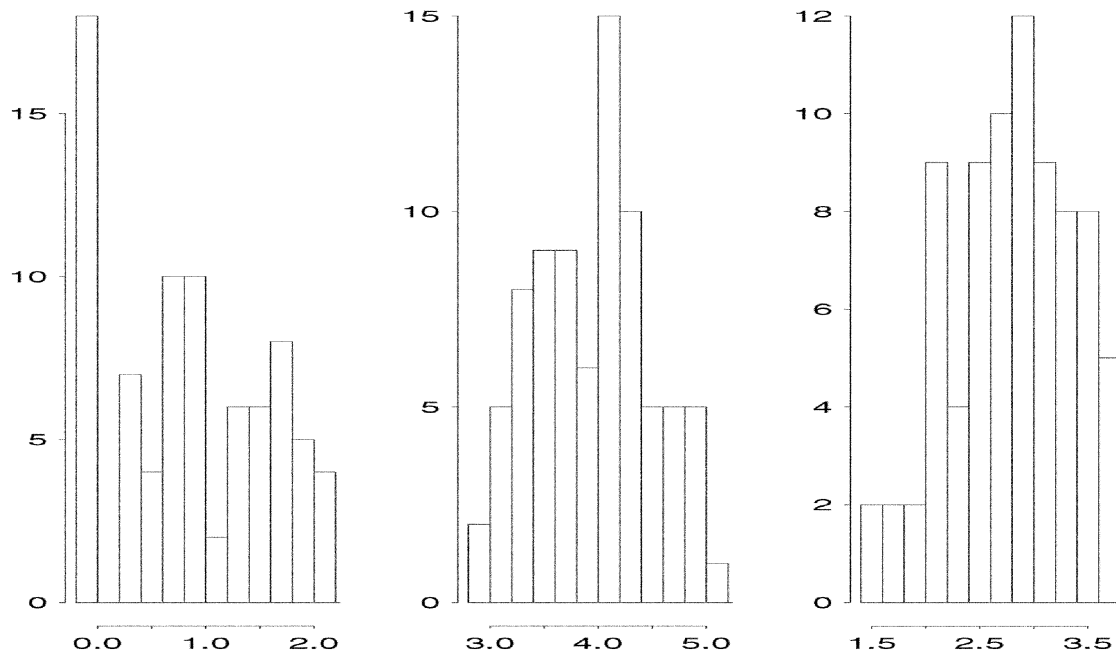$$\sum_{i=1}^{n}(e_i - y_i) + \sum_{i:y_i>0} y_i \log(y_i/e_i) \qquad (1)$$

Fig. 2. Histograms of various quantities for the modules under study. Left: histogram of base-10 log(1 + number of faults). Center: log (base 10) of lengths of modules as of the beginning of the prediction interval. Right: log (base 10) of numbers of deltas to the modules before the prediction interval.

afforded a compromise between the greater importance of large modules and their greater variability. This error measure is the deviance function for the Poisson distribution (that is, the estimated parameters minimize this function). By contrast, the least squares method would put too large a preference on models that did a good job of predicting the numbers of faults in unusually faulty modules. (In fact, the numbers of faults in our data are better described as having gamma distributions because their standard deviations tend to be proportional to their means, whereas Poisson modeling assumes that variances are proportional to means. Nevertheless, using the Poisson error measure provided a better trade-off between the greater importance and greater variability of faulty modules.)

3.2 Simulations to Assess Significance

To assess the amounts of uncertainty in parameter estimates and the statistical significance of differences between error measures of pairs of models, we ran simulation experiments. For example, suppose that we have fit a model with number of deltas and line age as predictor variables and that we wish standard error estimates for the coefficients, as well as to address whether length in lines adds materially to the model. To do this, we generated synthetic numbers of faults according to gamma distributions with means given by the model with deltas and age and with common shape parameters estimated from the data. We then computed the deviance for a model with deltas and age fit to the synthetic data and subtracted this deviance from the deviance for a model with deltas, age, and lines. We also recorded the

estimated parameters from the simulated data. After repeating this process a large number (2,000) of times, we can compute the standard deviation of the estimated parameters from simulated data and use this as an estimate of the standard error in the parameters estimated from the data. Also, we can count the number of times that the difference in deviance in the simulated data exceeded the change in deviance that the lines variable was responsible for in the data.

4 RESULTS

In this section, we present various models of modules' fault potential. First, we discuss the stable model, which predicts numbers of future faults using numbers of past faults. Next, we list our most successful generalized linear models, which construct predictors in terms of variables measuring the static properties of the code or comprising simple summaries of the change history; these variables are listed in Section 1.2.

4.1 Stable Model

This model simply predicts fault IMRs for each module in the two-year period beginning 1 April 1994, to be proportional to the number of fault IMRs in that module during the two-year period ending 1 April 1994. We refer to it as the stable model because it assumes that the fault generation dynamics remain stable across modules over time. This model provides no insight since it does not explain *causes* of faults, but it serves as a yardstick against which to compare other models. Improving upon it turns out to be fairly

TABLE 1
Models to Fit Fault Data

Model	Intcp	Common	Intl	US	Error
(A) Stable	-	-	-	-	757.4
(B) Null model	-	-	-	-	3108.8
(C) Organization only	3.46	0	-0.13	-1.39	2587.7
(D) $0.84\log(\text{lines}/1000)$	0.92	0	0.17	-0.92	1271.4
(E) $-0.14\log(\text{lines}/1000) + 1.19\log(\text{deltas}/1000)$	3.31	0	0.46	-0.70	980.0
(F) $1.05\log(\text{deltas}/1000)$	2.95	0	0.43	-0.72	985.1
(G) $0.07\log(\text{lines}/1000) + 0.95\log(\text{deltas}/1000) - 0.44\text{age}$	2.63	0	0.73	-0.65	696.3
(H) $1.02\log(\text{deltas}/1000) - 0.44\text{age}$	2.87	0	0.74	-0.63	697.4

difficult, however, because it implicitly incorporates effects for many of the predictors that might be used. After counting the fault IMRs for each module between 1 April 1992 and 1 April 1994, we renormalized the counts to have the same sum as the observed faults in the target interval, leading to an error (using (1)) of 757.4.

4.2 Generalized Linear Models

In all of our generalized linear models, we allowed three different intercepts—for international, domestic, and common modules. All other things being equal, international modules have four times as many fault IMRs as domestic modules, while common modules have twice as many as domestic. This does not, however, mean that international code is more likely to contain faults because of organizational differences in reporting practices: The international organization is known to report more IMRs for a given number of faults.

Table 1 lists the results of several different models for the log of the expected number of faults between 1 April 1994 and 1 April 1996. The predictor variables include $\log(\text{lines}/1000)$, the log of the number of thousands of lines of code in the module on 1 April 1994, comments included: $\log(\text{deltas}/1000)$, the (log of the) number of thousands of deltas to the module before 1 April 1994; and age, which measures the average age of the code in the module. Suppose that the deltas to the module occurred at dates $d_1, \ldots, d_n$, measured in years before 1990, and that the numbers of lines added to the module by these deltas are $a_1, \ldots, a_n$ respectively; then

$$\text{age} = \frac{\sum_{i=1}^{n} a_i d_i}{\sum_{i=1}^{n} a_i}.$$

Lines and deltas are measured in thousands and age in years before 1990 so that the intercept terms are of comparable orders of magnitude. Note that the log transformation of lines and deltas allows estimation of which power of these quantities best predicts numbers of faults, while age enters the model in an exponent so that we may ascertain the rate at which expected numbers of faults increase or decrease in the absence of new changes.

Each row of the table lists the form of the model (which variables it includes), the intercept, and the adjustment to the intercept necessary for the three organizational variables (modules changed by both the international and domestic organizations, those changed only by international, and those changed only by domestic). The parameterization of the model defined the adjustment to be zero

for the common modules. The final column shows the error measure associated with the model. The "Null" model predicts that every module will contain the same number of faults. The next model in the table, "Organization only," allows number of faults to differ depending on international, domestic, or both, but admits no other predictors. For instance, model F says that a common module with 5,000 deltas is predicted to have

$$\exp\{1.05\log(5000/1000) + 2.95\} = 104$$

faults during the period of interest. Similarly, model H predicts that a "U.S." module with 500 deltas and an average age of one year before 1990 will have

$$\exp\{1.02\log(500/1000) - 0.44 \times 1 + 2.87 - 0.63\} = 7$$

faults.

Several things stand out from the table. First, deltas are a much better measure of fault likelihood than lines and, second, once deltas have been taken into account, lines are not capable of improving the prediction. The contributions of deltas and age, however, appear to be quite important. The standard errors of the five coefficients in the last (deltas and age) model are, respectively, 0.17, 0.13, 0.19, 0.33, and 0.37; these were estimated using the simulation methodology discussed in Section 3.2. The coefficients of deltas are not statistically significantly different from one, which means that expected numbers of faults are proportional to numbers of deltas. The coefficient of -0.44 for age means that if one module's changes occurred a year earlier than those of another module with the same number of deltas and in the same branch, the older module will tend to have only $\exp(-0.44) \approx 0.64$ as many faults. This finding is consistent with our expectation that code which survives a long time is likely to be well-written.

To assess the statistical significance of differences between error measures of pairs of models, we ran a simulation experiment. We generated 2,000 differences in deviances as in Section 3.2 and compared them to the value of $697.4 - 696.3 = 1.1$ for the real data. Of the 2,000 sets of synthetic data, 1,713 of them (86 percent) had larger deviance differences than 1.1, so the improvement in the fit that the lines variable contributed in our data is small compared to the amount of natural variability in the problem. Thus, the data are consistent with a situation where lines add no predictive power beyond deltas and age.

It is less straightforward to answer the question of whether the model with deltas and age is better than the

TABLE 2
Correlations of Complexity Metrics

	1	2	3	4	5	6	7	8	9	10	11	12
1 Lines Of Code	1	.97	.88	.88	.91	.99	.98	.92	.97	.85	.72	.35
2 McCabe V(G)1	.97	1	.88	.90	.88	.95	.95	.89	.93	.86	.76	.29
3 Functions	.88	.88	1	.82	.89	.85	.84	.91	.84	.76	.65	.29
4 Breaks	.88	.90	.82	1	.83	.86	.85	.85	.85	.78	.67	.27
5 Unique Operators	.91	.88	.89	.83	1	.89	.87	1.00	.94	.65	.47	.48
6 Total Operands	.99	.95	.85	.86	.89	1	1.00	.90	.98	.85	.72	.31
7 Program Volume	.98	.95	.84	.85	.87	1.00	1	.88	.97	.87	.74	.28
8 Expected Length	.92	.89	.91	.85	1.00	.90	.88	1	.94	.69	.53	.42
9 Variable Count	.97	.93	.84	.85	.94	.98	.97	.94	1	.77	.60	.38
10 MaxSpan	.85	.86	.76	.78	.65	.85	.87	.69	.77	1	.92	-0.10
11 MeanSpan	.72	.76	.65	.67	.47	.72	.74	.53	.60	.92	1	-0.25
12 Prog Level	.35	.29	.29	.27	.48	.31	.28	.42	.38	-0.10	-0.25	1

stable model, but we can use similar simulations to get some idea of what constitutes a large difference in deviances. Of the 2,000 sets of simulated data, 284 had differences of $757.4 - 697.4 = 60$ or more, suggesting that the difference between the stable model and the model with deltas and age is within the range of natural variability. Rather than the negative conclusion that the model with deltas and age fails to improve the stable model, we interpret this to mean that a model suggesting causality (deltas cause faults) has the same explanatory power as (in effect) a model positing simply that the distribution of faults over modules is stationary over time.

4.3 Other Potential Predictors

One might wonder whether software complexity metrics (see, for example, [20]) are useful predictors of faults. We computed a number of metrics on the code as of 1 April 1996, including noncommentary source lines (NCSL), McCabe's single and full complexities [12], the numbers of functions, function calls, breaks and continues, unique operands and operators, total operands and operators; Halstead's program level, volume, difficulty, effort, and expected length [13]; and the mean and maximum spans of reference. (The results differ only slightly from the comparable values as of 1 April 1994 and were computed much more readily.) To get measures of complexity for modules, in most cases, we added over all the files in the module.

We found that nearly all of the complexity measures were virtually perfectly predictable from lines of code. (This analysis has excluded modules which contain only header and make files and, presumably, such modules would be very different in their complexities per line of code.) See Table 2 for a matrix of correlations of complexity meetrics.

This finding agrees in general with [15], in which most of the metrics with high correlations to faults had correlations of 0.9 or greater with each other. A notable exception in [15] was *SigFF*, a measure of how many signals a module sent to and received from other modules.

The failure of lines of code to be a useful predictor implies the same of the other complexity metrics. In fact, even using lines of code instead of noncommentary source lines does not impair the model performance. Within our data set, the correlation of $\log(1 + \text{total lines})$ and $\log(1 + \text{NCSL})$ was 0.9954.

Neither do unusually large values of complexity metrics *relative* to lines of code help predict how many faults a module has beyond typical modules with similar numbers of deltas: We compared ratios of metrics to NCSL (for the code on 1 April 1996) to the residuals from the GLM model and found no relationship.

We also tried relating the residuals from these models to other variables we thought might be related to fault potential. One hypothesis was that if a large number of developers had worked on a module beyond what the size and age of the module would predict, the code might be confused and, therefore, likely to contain faults. However, the number of developers was unrelated to the residuals, as was developers/deltas.

Also, developers with whom we are working believe that modules are complicated if they communicate with many other modules. We tried to measure this for a given module by taking the average number of other modules that were touched by MRs touching the module. This measure was also unrelated to the residuals, but we feel it is promising to consider other measures of connectivity based on the logical structure of the code.

4.4 Weighted Time Damp Model

Models that estimate a module's fault potential by adding an explicit contribution from each MR to the module are better than all of the previous models. In these models, an MR contributes a large amount of fault potential if it is large or if it is recent. Old changes either will have been fixed or will have been demonstrated to be fault-free.

These models begin by predicting the proportion of the total faults that given modules will have. We write $(e_1, \ldots, e_{80})$ as the unnormalized fitted fault potentials for the 80 modules, denote the times of the M MRs by $T_1, \ldots, T_M$, and the current time by t. Then, the models all have the form

$$e_i = \sum_{m=1}^{M} e^{-\alpha(t-T_m)} w_{im} \propto \sum_{m=1}^{M} e^{\alpha T_m} w_{im}, \qquad (2)$$

where w_{im} is the weight to the ith module corresponding to the mth MR. This model is qualitatively similar to the GLM with deltas and age, as it predicts high fault potentials for models with many recent deltas.

Several choices of the w's were examined, including

$$w_{im} = 1 \text{ if } m\text{th MR touches } i\text{th module, } 0 \text{ otherwise;} \quad (3)$$

$$w_{im} = \text{number of lines changed as part of this MR;} \quad (4)$$

and

$$w_{im} = \log(\text{number of lines changed as part of this MR}). \quad (5)$$

Lines changed is actually computed for a delta by adding the number of lines added to the number of lines deleted for that delta. A line is changed by deleting it and adding a replacement. Lines changed for an MR is the sum of lines changed over deltas within that MR. We also tried replacing numbers of lines changed by numbers of deltas in the weighting schemes, but we have had the most success with weighting (5) since it seems essential to take logs of quantities like lines or deltas when fitting models of this form.

The parameter α in (2) governs the rate at which the contribution of old MRs to the fault potential disappears. If α is large, only recent changes matter. On the other hand, the combination $\alpha = 0$ and w_{im}, as in (3), is very close to model F in Section 4.2, except that here we count MRs rather than deltas.

After evaluating the e_is, we renormalize them so that the sum of the predicted numbers of fault IMRs for the international modules is the same as the sum of the observed numbers of faults. We then repeat this process for the domestic and common modules.

We chose the log(lines) weighting scheme of (5). Minimizing the error measure over α leads to $\alpha = 0.75$ with an error measure of 631.0. This means that a change which is a year older than another change of the same size is only half as influential with respect to increasing fault potential since $\exp(-0.75) = 47\%$. This is similar to the coefficient of 64 percent fit in generalized linear model H above. The resulting error measure, 631.0, is slightly better than the 697.4 achieved by model H in Section 4.2. In the simulation experiment discussed in Section 3.2, only 256 out of 2,000 replications (13 percent) gave discrepancies larger than $66.4 = 697.4 - 631$, and only 84 out of 2,000 (4.2 percent) were larger than the difference between the deviances of the stable model and this model. Thus, there is strong evidence that this model is superior to the stable model and some evidence that it is also superior to the GLMs. It is interesting to see that treating changes individually, as in the weighted time damp model, leads to an improvement in the predictions.

Finally, we studied predictions over a different time interval, namely 1 October 1994 through 30 September 1995, the one-year period in the middle of the previous two-year prediction interval. The stable model—this time using fault IMRs from 1 October 1993 through 30 September 1994—attained an error measure of 570.2 and a GLM (with coefficients consistent with those fit to the two year data) achieved 557.9. Both of these models were decisively inferior to a weighted time damp model, this time with $\alpha = 12$, which gave an error measure of 350.1.

It is troubling that the two overlapping time intervals lead to values of α differing by an order of magnitude. For example, when $\alpha = 0.75$, changes made three months ago are $\exp(-3/16) = 83\%$ as influential as changes made

yesterday, while, when $\alpha = 12$, the corresponding factor is only $\exp(-3) = 5\%$. (However, for the one-year data, $\alpha = 0.75$ also is superior to the GLM, which has an error measure of 535.9.) The larger value of α appears to be the anomaly since the time interval October 1993 through September 1994 produces a value of α similar to that for the two-year data.

A bootstrap analysis (see [21]) of the two-year data concluded that the uncertainty associated with the value of α is large enough to suggest that 0.75 could plausibly be off by a factor of 2 in either direction (that is to say, a 95 percent bootstrap confidence interval for α would be $[0.375, 1.5]$), still leaving $\alpha = 12$ far outside. This remains disconcerting for those who would like to interpret α as a rate at which bugs are found and fixed.

5 SUMMARY

In this work, we developed several statistical models to evaluate which characteristics of a module's change history were likely to indicate that it would see large numbers of faults generated as it continued to be developed. Some of our conclusions are listed below.

- Our best model, the weighted time damp model, predicted fault potential using a sum of contributions from all the changes to the module in its history. Old changes were downweighted by a factor of about 50 percent per year.
- The best generalized linear model we found used numbers of changes to the module in the past together with a measure of the module's age. We obtained slightly less successful performance from these and from a model which predicts numbers of future faults from numbers of past faults.
- Models which account for number of changes made to a module could not be improved by including the module's length and, therefore, most software complexity metrics are of limited use in this context.
- We saw no evidence of a "too many cooks" effect: The number of developers who had changed a module did not help predicting numbers of faults.
- A measure of the frequency with which a module was changed in tandem with other modules was also a poor predictor of fault likelihood.

REFERENCES

[1] M.M. Lehman and L.A. Belady, *Program Evolution: Processes of Software Change.* Academic Press, 1985.

[2] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability.* McGraw-Hill, 1990.

[3] J. Jelinski and P.B. Moranda, "Software Reliability Research," *in Probabilistic Models for Software,* W. Freiberger, ed., pp. 485-502, Academic Press, 1972.

[4] G.J. Schick and R.W. Wolverton, "An Analysis of Competing Software Reliability Models," *IEEE Trans. Software Eng.,* vol. 4, no. 2, pp. 104-120, Mar. 1978.

[5] S.N. Mohanty, "Models and Measurements for Quality Assessment of Software," *ACM Computing Surveys,* vol. 11, no. 3, pp. 251-275, Sept. 1979.

[6] S.G. Eick, C.R. Loader, M.D. Long, L.G. Votta, and S. VanderWiel, "Estimating Software Fault Content Before Coding," *Proc. 14th Int'l Conf. Software Eng.,* pp. 59-65, May 1992.

[7] D.A. Christenson and S.T. Huang, "Estimating the Fault Content of Software Using the Fix-on-Fix Model," *Bell Labs Technical J.,* vol. 1, no. 1, pp. 130-137, Summer 1996.

[8] K.H. An, D.A. Gustafson, and A.C. Melton, "A Model for Software Maintenance," *Proc. Conf. Software Maintenance,* pp. 57-62, Sept. 1987.

[9] V.R. Basili and B.T. Perricone, "Software Errors and Complexity: An Empirical Investigation," *Comm. ACM,* vol. 27, no. 1, pp. 42-52, Jan. 1984.

[10] L. Hatton, "Reexamining the Fault Density–Component Size Connection," *IEEE Software,* pp. 89-97, Mar./Apr. 1997.

[11] T.J. Yu, V.Y. Shen, and H.E. Dunsmore, "An Analysis of Several Software Defect Models," *IEEE Trans. Software Eng.,* vol. 14, no. 9, pp. 1,261-1,270, Sept. 1988.

[12] T.J. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.,* vol. 2, no. 4, pp. 308-320, Dec. 1976.

[13] M.H. Halstead, *Elements of Software Science.* Elsevier–North Holland, 1979.

[14] N.F. Schneidewind and H.-M. Hoffman, "An Experiment in Software Error Data Collection and Analysis," *IEEE Trans. Software Eng.,* vol. 5, no. 3, pp. 276-286, May 1979.

[15] N. Ohlsson and H. Alberg, "Predicting Fault-Prone Software Modules in Telephone Switches," *IEEE Trans. Software Eng.,* vol. 22, no. 12, pp. 886-894, Dec. 1996.

[16] V.Y. Shen, T.-J. Yu, S.M. Thebaut, and L.R. Paulsen, "Identifying Error-Prone Software–An Empirical Study," *IEEE Trans. Software Eng.,* vol. 11, no. 4, pp. 317-324, Apr. 1985.

[17] J.C. Munson and T.M. Khoshgoftaar, "Regression Modelling of Software Quality: Empirical Investigation," *Information and Software Technology,* pp. 106-114, 1990.

[18] M.J. Rochkind, "The Source Code Control System," *IEEE Trans. Software Eng.,* vol. 1, no. 4, pp. 364-370, Dec. 1975.

[19] P. McCullagh and J.A. Nelder, *Generalized Linear Models,* second ed. New York: Chapman and Hall, 1989.

[20] H. Zuse, *Software Complexity: Measures and Methods.* Berlin, New York: de Gruyter, 1991.

[21] B. Efron and R.J. Tibshirani, *An Introduction to the Bootstrap.* New York: Chapman and Hall, 1993.

Todd L. Graves received the BS degree in statistics and probability from Michigan State University, East Lansing, in 1991, and the MS and PhD degrees in statistics from Stanford University, Stanford, California, in 1993 and 1995, respectively. He is a staff member of the Statistical Sciences Group at the Los Alamos National Laboratory. When this research was performed, he was a postdoctoral fellow at the National Institute of Statistical Sciences. His research interests include statistical analysis of software change management data, particularly modeling change effort, quality, and software modularity.

Alan F. Karr received BS (1969) and MS (1970) degrees in industrial engineering and PhD degree in applied mathematics (1973), all from Northwestern University. Currently, he is the associate director of the National Institute of Statistical Sciences, as well as a professor of statistics and biostatistics at the University of North Carolina at Chapel Hill. He is a fellow of the American Statistical Association and the Institute of Mathematical Statistics and a former member of the Army Science Board. His research interests include inference for stochastic processes, visualization and applications of statistics and stochastic models to transportation, materials science, software engineering, network computer intrusion, risk-limited disclosure of confidential data, and analysis of IP network data.

J.S. Marron received the BS degree in mathematics from the University of California at Davis in 1977 and the MS and PhD degrees in mathematics from the University of California at Los Angeles in 1980 and 1982, respectively. He is a professor of statistics at the University of North Carolina, Chapel Hill, and a senior researcher with the National Institute of Statistical Sciences, Research Triangle Park, North Carolina. He is associate editor for *Journal of the American Statistical Association*, *Journal of Nonparametric Statistics*, *Computational Statistics*, and *Test*. His research interests include statistical inference in smoothing methods, the analysis of populations of complex objects, and the statistical analysis of software change management data.

Harvey Siy received the BS degree in computer science from University of the Philippines in 1989 and the MS and PhD degrees in computer science from University of Maryland at College Park in 1994 and 1996, respectively. He is a member of the technical staff at Lucent Technologies. He is interested in conducting empirical studies to understand and improve the way large software systems are built and evolved.