

Adapting a Fault Prediction Model to Allow Widespread Usage

Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932
(weyuker,oststrand,rbell)@research.att.com

Abstract

In earlier work we investigated ways of predicting which files would be most likely to contain large numbers of faults in the next release of a large industrial software system. To make predictions we considered several different models ranging from a simple, fully-automatable model requiring no statistical expertise (the LOC model) to several different variants of a negative binomial regression model that were customized for the particular software system being developed. Not surprisingly, the custom models were invariably better at making the types of desired predictions than the simple model. However, customized models require knowledge about the tool used to record system changes, familiarity with the database formats used by this tool, as well as statistical expertise. We now extend our earlier research by presenting another large scale empirical study of the value of these prediction models using a new industrial software system over a nine year period. In addition, we introduce new models that are more sophisticated than the earlier LOC model, yielding more accurate predictions, but which nonetheless can be fully automated.

1 Introduction

In earlier research [4, 1], we developed and assessed prediction models to identify which executable files of a large multi-release industrial software system were likely to contain the largest numbers of faults in the next release. This information could then be used by testers to help them prioritize their testing efforts, and by developers to help them identify files that might need to be re-architected. We based our prediction model on a study [3] that looked at which characteristics of a large software system were most closely associated with files that turned out to be particularly problematic and used negative binomial regression models to make the predictions.

We first considered 17 releases of an inventory system,

representing more than 4 years of field release, and used our model to select what were predicted to be the 20% of the files that would contain the largest numbers of faults. These files in fact contained, on average, 83% of the faults.

For a service provisioning system (2+ years in the field), we observed the same results: the 20% of the files selected by the model contained 83% of the faults. Each of these systems contained close to 500,000 commented lines of code, and more than 2,000 files in the final release considered. Details of these results were described in [4].

Based on the success observed during these empirical studies, we considered three different variants of the negative binomial regression model for a third system, an automated voice response system. This system was particularly interesting to study because, unlike the two previous systems, it did not have regularly scheduled releases. Instead this project had what it referred to as “continuous releases.”

We found that the most accurate version of the prediction model was able to identify 20% of the system’s files that contained, on average, nearly three quarters of the faults that were actually detected later in the system’s usage [1]. This was particularly encouraging since we were able to get reasonably accurate results for a system whose development process was significantly different from the one for which the model was originally designed. Again this was a system containing several hundred thousand lines of code and close to 2,000 executable files. We followed this system and made predictions for nine successive synthetic quarterly releases.

While all these findings are very promising, model development for each system required extensive expertise and effort to specify the best form for predictors (e.g., transformations) and to select among predictors. This could discourage implementation of our methodology by practitioners for their systems. Furthermore, it may take a year or more to collect data on enough faults to support model development, thereby delaying the benefits. Despite these potential problems, we have found that although modeling results differ among the systems we have studied so far, the most important predictors tend to be common across systems.

This commonality suggests investigating whether most of the benefits can be attained using a model with a fixed, pre-specified set of predictors, where only the coefficients need be estimated on the fly. Although this approach would require software to fit negative binomial regression models, it would avoid the need for expert judgment about statistical modeling and would allow for quicker implementation. Going one step further, one might ask how effectively we could pre-specify a simple formula based on modeling done for previous systems, to completely bypass model development and to speed implementation to the greatest extent possible.

In this paper, we investigate both of these streamlined approaches and present the results of applying prediction models to a fourth large industrial system, with 35 successive releases produced over a period of roughly nine years. We designate this system the “maintenance support system”; it supports the maintenance processes for a number of products sold by a large international company. It has been developed and maintained by a different corporation from the one that produced the three systems that were the subjects of our earlier studies. Its size is roughly similar to the previous systems, with about 500,000 commented lines of code and approximately 700 executable files.

In addition to assessing the effectiveness of using some sort of prediction model for this new system, our goal in the work described here is to assess the degree to which it is feasible to design a uniform fault prediction model to determine files that are likely to require particular attention. The ultimate goal of this research is to produce an automated tool that could be used by testers and developers to identify the files most likely to be problematic in the future. In order for such a tool to be widely applicable, it should require little in the way of user statistical expertise, and require little human intervention or manual overhead.

Because of space limitations we omit a related work section here; however, the reader is directed to [1] for such a description and appropriate citations.

2 Model Requirements

Each of the systems considered to date has used the same commercially available integrated version control/change management system. Any change made to the system is requested and described in a *modification request* or MR.

Our approach has been to use the information in the MRs, and to base the predictions on objectively assessable factors including file size; whether the file was new to the system, in the sense that this is the first release in which it appeared; whether the file contained faults in earlier releases, and if so, how many; how many changes were made to the file in previous releases; and the programming language used to implement the file. The models developed

for the previous systems were customized based on observations of these factors made during the first few releases.

The purpose of the current research is to collect evidence indicating how effectively a standard prediction model can be devised for integration with an automated data extraction tool, so that little or no statistical expertise would be needed to produce fault likeliness predictions. The most extreme solution would involve the use of an entirely turnkey model for which model coefficients are standard and independent of the software system for which the predictions are being made. The coefficients would be based on results determined during our earlier studies. If the accuracy of the predictions were acceptably high, even if not as high as they were when using the customized prediction models, this might be a reasonable tradeoff: slightly reduced accuracy for automation.

Another alternative might involve adapting the model based on the new data extracted for a given new software system in some automatic or semi-automatic way.

The data extraction part of the fault prediction tool should perform its tasks without human intervention. For each of the previously analyzed systems, we wrote custom-designed scripts that extract the data needed for the prediction model from the MR database. Because all of the previous systems, as well as the current one, use the same MR system, we have recently implemented a common table-driven script that can be parameterized for each system. This script is the first part of an automated fault-proneness prediction system.

Of course, it is quite possible that other development projects will use different modification request systems, with different data storage formats. For each such change management system encountered, therefore, we plan to build a different back-end to the data extraction tool so that practitioners will only have to select the MR system used and the appropriate databases will be accessed to extract the needed data.

3 Data

For the maintenance support system, we analyzed data from 35 releases covering approximately nine years. Over that period, the number of executable files grew from 354 at the end of Release 1 to 668 at the end of Release 35. Intervals between releases generally ranged from three to five months, with the main exceptions being after Releases 1 (8.5 months), 2 (56 days) and 3 (40 days). Each release after the first one included a combination of files that were new to the system, along with ones that were new at a previous release. New files might be added any time during the life of a release, so we define a variable called *Exposure*, which is the fraction of the release for which a new file existed. Ten different languages contributed executable files to this

system, with the majority of files having been written in C or C++ (29% each).

Faults are identified from MRs that were classified as found by any of system test, customer test, user acceptance test, operational readiness test, or end-to-end test. A fault is defined as a change to an executable file in response to such an MR. There were a total of 1545 faults spread across the period. With the exceptions of four releases with fewer than 8 faults each, there were at least 18 faults in each release. MRs also provided information about the total number of changes to executable files, for all purposes, during each release.

4 Models Assessed in this Paper

As in [4, 1], we use negative binomial regression [2] to model the number of faults in a file during a specific release. The unit of analysis is the combination of a file and release. Negative binomial regression models the logarithm of the expected number of faults as a linear combination of the explanatory variables. We estimated model parameters by maximum likelihood, with all computations performed using Version 9.1 of SAS [5]. A fuller description of this methodology can be found in [1]. Once a model has been estimated, we compute predicted numbers of faults for files in the next release to prioritize files for purposes of testing.

We excluded Release 1 from our modeling because it appeared to differ systematically from later releases in ways that might make its results unrepresentative of what followed. As noted above, that release lasted twice the norm. In addition, there was a very high rate of file changes, but only 22 changes were identified as faults, because almost no system testing occurred during that release.

We assess four distinct models/algorithms for the maintenance support system. The four alternatives differ in the degree of pre-specification. The first two are fully specified formulas, including coefficients. The third is a pre-specified model, whose coefficients are to be estimated from data for the current system. The fourth is an open-ended “algorithm” for development and estimation of model. The four alternatives are:

1. *Lines of code (LOC) only.* In this model, the files are simply ordered from longest to shortest, and file length is the sole factor determining the prediction of which files are most likely to contain large numbers of faults. This option was included because we have found in our earlier studies that the size of the file was generally the most important characteristic for determining fault-proneness.
2. *Pre-specified formula, including coefficients.* The files are ordered by a linear combination of four variables.

These variables and their coefficients are shown in Table 1.

For this model, both *New file* and *Changed file* are dummy variables. For *New files*, exposure is the proportion of time during the release that the file exists. Exposure is always 1 for *Old files*. The coefficients are set at approximately the average of values estimated from the inventory system and automated voice response systems studied in earlier research as described in [4, 1], where applicable. This model was limited to predictors judged likely to be essential, and for which coefficients could be pre-specified with moderate confidence. We do not specify an intercept term because one would not affect the ordering of files that is needed for our assessment.

3. *Pre-specified model, coefficients estimated from data.* In this case the coefficients to predict Release N are based on a model fit to Releases 2 to (N-1). The set of variables used are: log(KLOC); dummy variables for New files (Age = 0), Age = 1, and Age = 2-4 (where applicable), with Age > 4 as the reference set; the logarithm of Exposure; square roots of Changes in the prior release, Changes two releases ago, and Faults in the prior release; dummy variables for selected programming languages; and dummy variable for all but one release. To reduce the potential of overfitting for less prevalent programming languages, we included in the model only those languages with a cumulative total of 20,000 LOC (20 KLOC) across releases, with an average of at least 2 KLOC per release in recent releases. This allowed dummy variables for files written in C or C++ immediately and for four additional languages after a few releases.
4. *Customized model based on Releases 2 to 5.* In this case, exploratory analysis allows evaluation of alternative transformations, variable selection, and (possibly) interactions.

Because Models 1 and 2 are completely pre-specified, we are able to start predictions with the first normal release, Release 2, when these models are used. In contrast, because the coefficients for Model 3 need to be estimated from data for at least one prior release, we begin predictions with Release 3. For Model 4, the first predictions are delayed until Release 6 because model development calls for additional training data (Releases 2 to 5).

5 Findings

Our development of a customized model (Model 4) for the maintenance support system produces a model very similar to the pre-specified Model 3. Modeling faults for

Variable	Coefficient
log(KLOC)	0.7
New file	2.0
Changed file	1.0
log(exposure)	0.6

Table 1. Coefficients for Pre-specified Model 2

SYSTEM	LOC model	CUSTOMIZED model
Inventory	73%	83%
Provisioning	74%	83%
Voice Response	56%	75%

Table 2. Percentage of Faults in Top 20% of Files for Previously Studied Systems

Releases 2 to 5 confirmed the importance of log(KLOC), log(Exposure), and dummy variables for release number. This analysis also confirmed much higher fault rates for new files. In contrast with Model 3, exploratory analysis suggested a dummy variable for files with Age equal to 1 or 2; we note, however, that it is difficult to infer the appropriate way to handle file age with only a few releases worth of data. Comparison of log likelihoods between models using alternative transformations for prior changes and prior faults suggested that square root transformations were preferable to either raw counts or indicators of “any” versus “none” for both variables. In contrast to Model 3, exploratory analysis led to the exclusion of the count of changes two releases ago in any form. The same analysis led to the inclusion of dummy variables for two programming languages, which we denote L1 and L2, neither of which is C or C++.

Finally, certain unexpected findings for Release 2 (e.g., there were no faults for new files) suggested that Release 2 may not be predictive of what follows. Consequently, for Model 4, we chose to use Releases 3 to (N-1) as the training set for Release N. For example, using data for Releases 3 to 20 yields the following formula for the expected number of faults at Release 21:

$$E(\text{Faults}) = \exp [2.42 + 0.53 \cdot \log(\text{KLOC}) + 1.68 \cdot \text{New} + 0.17 \cdot (\text{Age} = 1 \text{ or } 2) + 1.14 \cdot \log(\text{Exposure}) + 0.72 \cdot \sqrt{\text{Prior Changes}} + 0.38 \cdot \sqrt{\text{Prior Faults}} + 1.18 \cdot \text{L1} + 0.86 \cdot \text{L2}],$$

with estimated dispersion parameter of 1.75.

For comparison purposes, we provide a summary of our prediction results from the earlier systems studied. For each of those systems, Table 2 shows the average percentages of faults appearing in the 20% of files predicted to have the most faults based on either LOC or a model customized for the particular system.

Table 3 summarizes prediction results for the maintenance support system studied in this paper. Using groups of

five releases, we report the percentage of faults contained in the top 20% of files as ordered by the four models. The last two rows show mean values for each of the models over all releases through Release 35, as well as the mean values for each of the models over Releases 6 through 35. The reason we considered that restricted range is that data from the first 5 releases were used as training data for Model 4.

The results for Model 3 (the pre-specified model) and Model 4 (the customized model) are consistently at or near the best of the four models—with each exceeding 80% of faults on average. Both models tended to improve over time as the amount of training data increased. Overall, for the releases in common, Model 3 did slightly better. This is a very encouraging observation since Model 3 is one which we expect to be able to fully automate. We also see from this table that Model 3 is comparable to the results obtained using a customized model for the inventory system, and better than the results observed for the voice response system (Table 2).

Model 2 trails Models 3 and 4, but still reaches 75% across all releases. To investigate whether the shortfall relative to Model 3 is due more to Model 2 using a much shorter list of predictor variables or to the pre-specification of the coefficients (Table 1), we also assessed a version that predicts faults at release N using coefficients estimated from the same model fit to data for Releases 2 to (N-1). Those results suggest that most of the shortfall is due to the pre-specification of coefficients.

We had originally introduced the Lines of Code Model because it requires no expertise to compute; one only needs to sort the files based on their size and select the biggest files for consideration. We saw that although it was less successful than the fully custom prediction model, it nonetheless did far better than would be expected if one randomly selected the order of files to be tested. However, the LOC model consistently falls short of the other options, especially after about Release 15.

Release Number	Model 1	Model 2	Model 3	Model 4
2-5	50	58	55	NA
6-10	63	66	78	77
11-15	56	66	71	68
16-20	68	80	84	81
21-25	77	91	89	91
26-30	72	83	90	87
31-35	63	79	92	92
Mean for all Releases	64.4	75.2	81.3	82.6
Mean for Rel 6-35	66.3	77.4	83.9	82.6

Table 3. Percentage of Faults in Top 20% of Files, by Model for the Maintenance Support System

6 Conclusions

The results for this system generally confirm those from the three systems studied previously. For the best models, 20 percent of files yield more than 80 percent of faults, averaged across releases. Also, the same predictor variables generally led the way.

We were able to avoid model development (definition of predictors and variable selection) and still estimate effective prediction models. Indeed, Model 3 slightly outperformed Model 4, developed specifically for this case study. In other words, several years of data for other systems was as least as informative as 10 months of data for the current system. Furthermore, the even simpler pre-specified formula (Model 2)—which bypasses model estimation as well as model development—performed surprisingly well, losing only about six percentage points across releases (relative to Model 3).

In contrast, the LOC model loses quite a bit relative to both Model 2 (11 percentage points) and to Model 3 (17 percentage points), especially at later releases when the main discriminators are the status of files as either *new* or *changed*. Given the improvement possible with very little marginal effort, we recommend against the LOC-only model.

It is risky to generalize too much from a small number of systems. How well any particular model can predict faults may depend on many system attributes, including size, complexity, the mix of programming languages, the rate of additions and modifications, and testing style. Such factors may play an even larger role in whether models developed on one set of systems translate well to others.

Nonetheless, the strong consistency across the systems we have studied provides promising evidence that our methodology can be implemented in the real world without extensive statistical expertise or modeling effort. Results for Models 2 and 3 indicate that a prediction model can be both simple and highly accurate.

References

- [1] R.M. Bell, T.J. Ostrand, and E.J. Weyuker. Looking for Bugs in All the Right Places. *Proc. ACM/International Symposium on Software Testing and Analysis (ISSTA2006)*, Portland, Maine, July 2006, pp. 61-71.
- [2] P. McCullagh and J.A. Nelder. *Generalized Linear Models*, Second Edition, Chapman and Hall, London, 1989.
- [3] T. Ostrand and E.J. Weyuker. The Distribution of Faults in a Large Industrial Software System. *Proc. ACM/International Symposium on Software Testing and Analysis (ISSTA2002)*, Rome, Italy, July 2002, pp. 55-64.
- [4] T.J. Ostrand, E.J. Weyuker, and R.M. Bell. Predicting the Location and Number of Faults in Large Software Systems. *IEEE Trans. on Software Engineering*, Vol 31, No 4, April 2005.
- [5] SAS Institute Inc. *SAS/STAT 9.1 User's Guide*, SAS Institute, Cary, NC, 2004.