An Empirical Comparison of Field Defect Modeling Methods

Paul Luo Li, Mary Shaw, Jim Herbsleb,
P. Santhanam*, Bonnie Ray*
May 2005
CMU-ISRI- 06-102

Institute for Software Research International
School of Computer Science
Carnegie Mellon University
Pittsburgh PA, 15213

*Center for Software Engineering
IBM T.J. Watson Research Center
Hawthorne, NY 10532

This research was supported by the National Science Foundation under Grants ITR-0086003 and CCF-0438929, by the Carnegie Mellon Sloan Software Center, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. This research is possible through an IBM joint study agreement.

Keywords: Empirical Studies, Metrics, Reliability Engineering Management, Measurement, Reliability, Experimentation, Defect modeling, empirical research, COTS, maintenance resource planning, software insurance

ABSTRACT
In this study, we report empirical results from forecasting field defect rates and predicting the number of field defects for a large commercial software system. We find that we are not able to accurately forecast field defect rates using a combined time-based and metrics-based approach, as judged by the Theil forecasting statistic. We suggest possible conditions that may have contributed to the poor results. Next, we use metrics-based approaches to predict the number of field defects within the six months after deployment. We find that the simple ratios method produce more accurate predictions than more complex metrics-based methods. Our results are steps toward quantitatively managing the risks associated with software field defects.

1 INTRODUCTION

The US Department of Commerce estimates that software field defects cost the U.S. economy an estimated $59.6 billion dollars annually and that over half of the costs are borne by software consumers and the rest by software producers [24].

Field defect modeling may enable methods that mitigate the risks associated with field defects for software producers and software consumers. Field defect modeling may help manage the risks by guiding testing [9], improving maintenance resource allocation [15], adjusting deployment to meet the quality expectations of customers [22], planning improvement efforts [2], and enabling a software insurance system [19].

In this paper, we report empirical results from modeling field defects for twelve releases of a commercial operating system. The specific questions addressed by this paper are:

1. Can a combined metrics-based and time-based forecasting approach be used to accurately forecast field defect rates?
2. Which metrics-based modeling method produces the most accurate predictions of the number of field defects in the first six months after deployment?

Li et al. [15] has shown that it is possible to accurately forecast field defect rates before release using a combined metrics-based and time-based method for an open source operating system. The authors predict the parameters of Software Reliability Growth Models (SRGMs) using metrics-based modeling methods. In this paper, we attempt to predict field defect rates for a commercial operating system. However, we find that the same approach does not yield accurate forecast for a commercial operating system as judged by the Theil forecasting statistic (explained in section 4). We conjecture about the conditions under which accurate field defect rate forecasts using the combined time-based and metrics-based approach is not possible.

Next, we predict the number of field defects in the first six months after deployment using metrics-based methods from prior work. The predicted number of field defects in the first six months after deployment may allow assessment of the initial quality of the software system and may enable initial maintenance resource planning. We compare accuracy of predictions and find that the simple ratios method produces more accurate predictions than other methods based on absolute relative error (explained in section 4). We conjecture that the maturity of the software development organization may have contributed to the superiority of the ratios method.

Section 2 provides related work and motivation for our work. Section 3 provides background information on the commercial operating system. Section 4 provides background on the SRGMs, metrics-based methods, and techniques for evaluating our results. Section 5 presents the results and discussions.

2 MOTIVATION AND RELATED WORK

In this section we will discuss how previous work motivates our study. Discussion of the SRGMs and metrics-based modeling methods are deferred to section 4.

2.1 Types of prediction

Predictions regarding field defects in prior work generally belong to one of four categories:

- Relationships: These studies establish relationships between predictors and the field defects. For example, Harter et al. [6] establish a relationship between an organization's CMM level and the number of field defects in projects completed by the organization.

- Classifications: These studies predict if the number of field defects is above a threshold for a given observation. For example, Khoshgoftaar et al. [9] classify modules as risky (will contain at least one field defect) or not risky (no field defects) for changed modules.
- Quantities: These studies predict the number of field defects. For example, Khoshgoftaar et al. [13] predict the number of defects for modules of two software systems.
- Rates of occurrences over time: These studies predict the field defect rate. For example, Kenny [8] predicts the field defect rates captured by the Weibull model for two IBM systems.

In this paper, we first attempt to predict the rate of field defects. Field defect rate forecasts answer how many field defects will occur and how the field defects will be distributed over time. This type of prediction is preferred for methods that mitigate the risks associated with field defects as discussed by Li et al. in [18].

Since we are unable to predict the rate of field defects accurately, we attempt to predict the number of field defects in the first six months after deployment. This prediction may allow decision makers to make initial plans.

2.2 Prediction approaches

Field defect predictions generally belong to one of two classes: time-based approach and metrics-based approach. Schneidewind [27] distinguish between these two approaches:

1. Time-based approach: This approach uses defect occurrence times or the number of defects in time intervals during testing to fit a software reliability model. The rate of field defects is estimated using the fitted software reliability model. Lyu [20] describes this approach in detail.
2. Metrics-based approach: This approach uses historical information on metrics available before release (predictors) and historical information on field defects to fit a predictive model. The fitted model and predictors' values for a new observation are used to make predictions. Examples of this approach are in Mockus et al. [22] and Khoshgoftaar et al. [9].

In order for the defect occurrence pattern to continue from testing into the field, the software has to be operated in a similar manner as that in which reliability predictions are made. The similarity of testing and deployment environments assumption is one of the key assumptions for the time-based approach cited by Farr in [20]. However, we are interested in widely-used systems such as COTS and open source software systems. The similarity of testing and deployment environments assumption does not necessarily hold or these systems. Therefore, it may not be appropriate to forecast field defect rates using a software reliability model fitted using testing information.

Unlike the time-based approach, the metrics-based approach uses historical information on predictors and actual field defect information to construct a predictive model. Since there is no assumption about the similarity between testing and deployment environments, metrics-based models are more robust against differences between how the software is tested and how it is used in the field.

Prior work has used the metrics-based approach to predict relationships (e.g. Harter et al. [6]), classifications (e.g. Khoshgoftaar et al. [13]), and quantities (e.g. Mockus et al. [22]); however, only no previous study forecasted the rate of field defects.

Li et al. [15] forecast field defect rates using a novel method that combines metrics-based and time-based approaches. The authors use metrics-based modeling methods to predict model parameters of SRGMs, which captures the rate of field defects. Results show that accurate predictions are possible. In this paper, we first use the combined time-based and metrics-based approach to predict the rate of field defects, and then we use metrics-based modeling methods to predict the number of field defects in the first six months after release.

Khoshgoftaar and Seliya have compared the prediction accuracy of several metrics-based methods in [15]. We differ from Khoshgoftaar and Seliya in three ways. First, we predict for entire software releases, where as Khoshgoftaar and Seliya predict for software modules. Secondly, we use a real-time evaluation technique (explained in section 4) to compare methods. Finally, we consider three additional methods: the moving averages method, the exponential smoothing method, and the ratios method.

2.3 Categories of metrics

Metrics-based prediction methods require metrics. Metrics available before release are *predictors,* which can be used by metrics-based modeling methods.

We categorize predictors used in prior work using an augmented version of the categorization schemes used by Fenton and Pfleeger in [5] and Khoshgoftaar and Allen [9]:

- Product metrics: metrics that measure the attributes of any intermediate or final product of the software development process. Product metrics have been shown to be important predictors by studies such as Khoshgoftaar et al. [9].
- Development metrics: metrics that measure attributes of the development process. Development metrics have been shown to be important predictors by studies such as Mockus et al. [21].
- Deployment and usage metrics (DU): metrics that measure attributes of deployment of the software system and usage in the field. DU metrics have been shown to be important predictors by studies such as Jones et al. [7].
- Software and hardware configurations metrics (SH): metrics that measure attributes of the software and hardware systems that interact with the software system in the field. SH metrics have been shown to be important predictors by Mockus et al.[22].

Li et al. [15] uses predictors in each category (145 predictors in all) to predict the rate of field defects; however, the authors suggest that predictions may be possible using fewer metrics. In this study, we attempt to make predictions using only four development metrics.

3 DATA AND SYSTEM DESCRIPTION

The operating system we examine is developed by IBM and is a mature product with many years of presence in the marketplace.

The defect-occurrence data collected are code-related problems discovered and reported by customers after deployment. The defect-occurrence data for the operating system contain unique field defects that led to code changes by the development organization.

The defect-occurrence data provided to us are pre-processed and aggregated, so we use the interval in the data set. The time interval for the operating system is a quarter (i.e. 3 months).

In addition to field defect occurrence data, we also have aggregated change information and development defect information. For each release, we have the total lines of code changed relative to the previous release and the number of in-process (i.e. development) defects.

Using the field defect occurrence data and lines of code-changed data, we also compute two derived metrics. We compute the number of field defects in the previous release observed during the development period. This metrics is used by Khoshgoftaar et al. in [11]. We computed the ratio of the amount of code changed in the current release to the amount of code changed in the previous two releases. This is similar to the relative code churn metric used by Nagappan and Ball in [23]. The metrics used are summarized in table 1.

Table 1. Predictors

Metric	*Description*
Changed lines	Lines of code changed relative to the previous release
Dev defects	Number of in-process (i.e. development) defects
Prev defects	Number of field defects in the previous release observed during the development period of the current release
Ratio changed	Ratio of the amount of code changed in the current release to the amount of code changed in the previous two releases

4 PREDICTION AND ANALYSIS METHOD

In this section, we describe the SRGMs used to model the field defect rates, the metrics-based modeling methods, and the techniques for evaluating forecasts and predictions.

In this paper, we simulate a real world situation by making forecasts and predictions in the first six months after release using only information available at the time of release for multiple releases.

Prior work either inadequately addresses multiple releases or does not account for multiple active releases. Some studies (e.g. Khoshgoftaar et al. [13]) split data from the same release into fitting and testing sets. This approach ignores possible differences between releases that are not accounted for in the model. Some studies (e.g. Ostrand et al. [25]) use a model fitted using data from a historical release to predict for future releases. However, this approach assumes that complete field defect information is available for historical releases; yet, complete field defect information is often not available for historical releases that are still active in the field.

When predicting the rate of field defects, we estimate model parameters for active historical releases using only information available at the time of release. This is the same approach taken in Li et al. [15]. When predicting the number of field defects in the first six months after release, we use information from historical releases that have complete defect information (i.e. have been in the field for more than six months) at the time of release.

4.1 Software reliability growth models

Li et al. [18] have compared the ability of SRGMs to model the rate of field defects of the commercial operating system. Based on post-facto fits the authors conclude that the Weibull model is superior to other models.

However, the results in Li et al. [18] are based on post-facto fits. Li et al. [15] suggest that Weibull model parameters may be harder to predict compared with model parameters of simpler models (e.g. the exponential model). Therefore, we also consider the Gamma model and the Exponential model, which have been shown to be the next most effective models at modeling field defect rates. The models forms are in table 2. The model parameters (in bold) dictate the rate of field defects. To forecast field defect rates, we use metrics-based modeling methods to predict these model parameters.

Table 2. Software reliability models

Model type	*Model form*
Exponential	$\lambda(t) = \mathbf{N}\,\boldsymbol{\alpha}\, e^{-\alpha t}$
Weibull	$\lambda(t) = \mathbf{N}\,\boldsymbol{\alpha}\,\boldsymbol{\beta}\, t^{\alpha-1} e^{-\beta t^{\alpha}}$
Gamma	$\lambda(t) = \mathbf{N}\,\boldsymbol{\beta}^{\alpha}\, t^{\alpha-1} e^{-\beta t}$

4.2 Metrics-based modeling methods

Prior work (e.g. Jones et al. [7], Ostrand et al. [25], Khoshgoftaar et al. [11], Khoshgoftaar and Seliya [14]) has explored using various metrics-based modeling methods to predict quantities (e.g. the total number of field defects). Given our data constraint, some of the methods used in

previous studies may not be appropriate (e.g. there is not enough training data); however, for completeness we attempt each method. We discuss the methods and our modifications.

4.2.1 Principle component analysis, clustering, and linear regression

We roughly replicated (explained below) the principle component analysis (PCA), clustering, and linear regression method in Khoshgoftaar et al. [12] .

Khoshgoftaar et al. [12] first constructed principle components and then clustered observations based on the principle components. Finally, linear models were fitted to the observations in each cluster. To predict for a new observation, the observation was placed into one of the clusters based on its predictors' values. The fitted linear model for the cluster was then used to predict the value of the new observation.

Khoshgoftaar et al. [12] predicted field defects for modules using 11 product metrics. The authors used 260 observations to fit the model and used four clusters. Since we had at most 11 observations, we modified the process to use two clusters and to fit a null linear model (i.e. an average of the observations) for each cluster. In addition, we did not have enough observations to perform a PCA for most of the releases; therefore, we did not perform a PCA. We used the popular K-means clustering method, since the referenced work did not identify the clustering method used.

4.2.2 Linear regression with model selection

We replicated the linear regression with model selection method in Khoshgoftaar et al. [13]. The idea behind linear regression is that changes in a predictor's value changes the predicted quantity by a fixed amount. Model selection balances the bias-variance trade-off by including only predictors that have the most amount of benefit or by dropping predictors that have the least amount of benefit as judged by a model selection criterion (e.g. AIC).

Khoshgoftaar et al. [13] used backwards and stepwise model selection techniques to select a subset of predictors. A linear regression model was fitted using the selected predictors and the least squares method. To predict for a new observation, the predictors' values and the fitted model were used to estimate the value.

Khoshgoftaar et al. [13] predicted field defects for modules of two systems using 8 product metrics for one system and 11 product metrics for the other system. The authors work used 188 and 226 observations to fit models for the two systems. Due to data constraints, we modified our model selection method to select only one predictor to prevent over fitting. Since no model selection criterion was identified in the paper, we used the popular AIC model selection criterion.

4.2.3 Non-linear regression

We replicated the non-linear regression method used in Khoshgoftaar et al. [10]. The idea behind non-linear regression is that changes in predictor's value change the predicted value by a parameterized amount.

Khoshgoftaar et al. [10] constructed a non-linear model using non-linear least squares regression of the form:

$y = b_0 + b_1 * (LOC)^{b2}$

y = number of faults, b_0, b_1 , b_2 were modeling parameters, LOC was lines of code

For a new observation, the value of the lines of code metrics was inserted into the model to produce a prediction.

The authors used 15 observations to train the model. We found that it was not possible to fit a model with three parameters; so, we simplified the model by dropping a parameter. Our model was:

$y = b_1 * (LOC)^{b2}$

4.2.4 Trees

We replicated the Classification and Regressions Trees (CART) method in Khoshgoftaar and Seliya [14]. The idea behind trees is that predictors have critical values that distinguish between similar observations and that all similar observations have similar predicted values.

Khoshgoftaar and Seliya [14] first built a regression tree using training observations and a minimum node size before further splitting of 10. To predict for a new observation, the observation traversed the tree according to its predictors' values until it reached a leaf. The mean of the predicted value of the training observations in the node was the predicted value of the new observation.

Khoshgoftaar and Seliya [14] predicted field defects in modules using 9 product metrics. The model was fitted using 4648 observations. Since we had at most 11 training observations, we built trees with varying minimum node sizes of between 2 to 7.

4.2.5 Neural networks

We replicated the feed-forward neural networks method used in Khoshgoftaar et al. [13]. The idea behind neural networks is that predictors' values are like neural inputs, which can be used by a neural network to arrive at a conclusion about a new observation.

A neural networks model is a multi-layer perceptron model that produces a value between 0 and 1. The predictors are in one layer, with each predictor as one neuron, and the output is in one layer. There is at least one intermediate hidden layer in-between with different number of neurons. Each neuron in one layer is connected to each neuron in the next layer. The connection strength between the neurons can vary. A non-linear function is used to combine values coming into the neuron to produce the output from the neuron. For a new observation, the predictors' values are placed on the outer layer and the predicted value between 0 and 1 is produced at the output neuron.

Khoshgoftaar et al. [13] scaled all values (predictors and the predicted value) to be between 0 and 1 by dividing by the value of the maximum element in each set. The data were then used to fit a neural network trained using backward error propagation. To predict for a new observation, the predictors' values were used to produce a value between 0 and 1. The value was then scaled up according to the range of the predicted value in the training set.

Khoshgoftaar et al. [13] predicted field defects for the same two systems as the linear regression with model selection method. The authors used 16 and 18 hidden layer neurons for the two systems. We modified the process by fitting separate neural networks for each predictor (i.e. one input neuron) using one hidden layer neuron. For each release, we selected the preferred model by evaluating predictions for the fitting set using each fitted model. The most accurate model was used to fit the make predictions for the next release.

4.2.6 Exponential smoothing and moving averages

We replicated the moving averages and exponential smoothing methods used in Li et al. [18]. The idea is that past releases are similar to the current release.

To predict for the next release, a weighted average of the values from historical releases was used. For the moving averages method, each historical release received equal weight. For exponential smoothing method, releases closer in time received more weight. The idea was that recent releases were more similar to the current release. Li et al. [18] considered averaging 2-7 releases. We made no modifications to the method; however, due to data shortages we considered averaging 2-6 releases.

We also added a reverse smoothing method in which releases that were further away in time receive more weight. The logic behind this method was that due to data availability, releases that

were further away in time had more stable and accurate estimated model parameters; therefore, those releases should receive more weight.

4.2.7 Ratios

We extended prior research in Biyani and Santhanam [3]. The authors found that information on the most recent release was sufficient to predict defect volume in the subsequent release for software modules. In addition, the authors determined that the ratio of field defects to development defects could be used as an accurate indicator of quality for individual modules. We used ratios to predict the number of field defects.

To predict for the next release, we first found the ratio between the number of field defects and the number of development defects in the most recent release. We then used this ratio and the number of development defects in the current release to predict the number of field defects for the current release.

4.3 Methods of evaluation

In this section, we will first discuss how we evaluate forecasted rate of field defects then we will discuss how we evaluate predicted number of field defects.

All analysis was preformed using the open source analysis package R [26].

4.3.1 Evaluating predicted field defect rates

In this study, model parameters of SRGMs were predicted using one of the metrics-based modeling methods (the same method for all model parameters). Each forecast was evaluated using the Theil forecasting statistic.

The Theil statistic compares the forecast for each time interval *i* against a no-change forecast based on the previous time interval's value [28].

$$U^2 = \frac{\Sigma (P_i - A_i)^2}{\Sigma A_i^2}$$

The Theil statistic U is greater or equal to zero. The term P_i is the projected change and A_i is the actual change in interval i. A Theil statistic of zero indicates perfect forecasts with $P_i = A_i$. A Theil statistic of one indicates that forecasts are no better than no-change forecasts with $P_i = 0$. Values greater than 1 indicate forecasts are worse than no-change forecasts. We consider forecasts accurate if the resulting Theil statistic is less than 1.

4.3.2 Evaluating predicted number of field defects in a specific time interval

Each prediction was evaluated using the Absolute Relative Error (ARE). The absolute average error (AAE) and the ARE are the most commonly used measures of accuracy for predicted number of field defects. The AAE measures the average error in predictions (i.e. how much a typical prediction is off by). The ARE measures the average percentage of error in the predictions (i.e. relative to the actual number of field defects, how much a typical prediction is off by). We use ARE for two reasons. First, the AAE can be misleading when the predicted number of field problems differs significantly between observations. Second, we use ARE to preserve confidentiality.

Absolute relative error is defined by Khosghgoftaar et. al. in [13] as the sum over all observations, the absolute value of the difference between the predicted value and the actual value divided by the actual value.

$$\text{ARE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\overline{y}_i - y_i}{y_i}\right|$$

5 RESULTS AND DISCUSSION

In this section we present results of our experiments. We also discuss the conditions that may have contributed to the results.

5.1 Field defect rate forecasts

Table 3 contains the top ten most accurate forecasts based on average Theil statistics. Column one contains a description of the metrics-based modeling method and SRGM combination used, column two contains the average Theil statistic achieved by the combination, column three contains the range of the Theil statistics (i.e. the difference between the worst and the best Theil statistic), and column four contains the number of releases that the combination predicted for. Figures 1-3 present sample predictions. All predicted defect occurrence patterns are plotted for each release. The lines in the figures are forecasts that are closest to actual defect occurrence patterns.

Table 3. Theil statistics

Method	*Average Theil*	*Range*	*Releases predicted*
Weibull, reverse smoothing 6 releases	1.0	0.6	2
Weibull, exponential smoothing 6 releases	1.4	0.0	2
Gamma, moving average 4 releases	1.6	3.3	3
Weibull, moving average 4 releases	1.7	3.5	3
Gamma, exponential smoothing 4 releases	1.9	3.8	3
Gamma, exponential smoothing 5 releases	1.9	3.8	3
Exponential, linear regression	1.9	2.7	5
Weibull, exponential smoothing 4 releases	2.0	4.1	3
Weibull, exponential smoothing 5 releases	2.0	4.1	3
Gamma, clustering	2.1	3.5	7

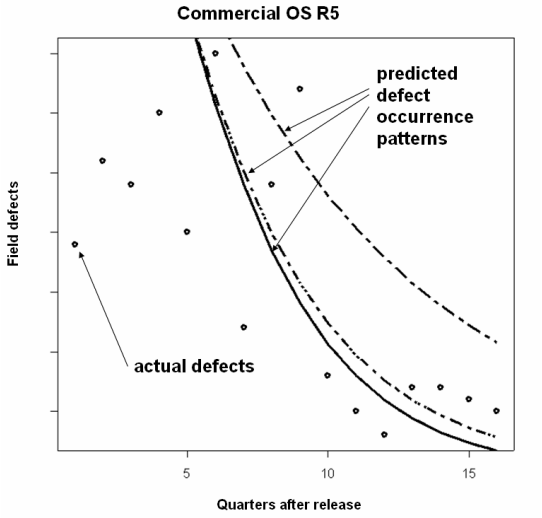

Figure 1. Actual and predicted field defects for OS Release 5

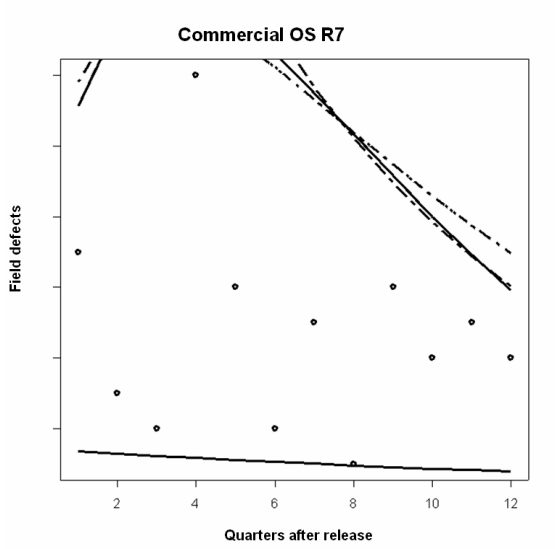

Figure 2. Actual and predicted field defects for OS Release 7

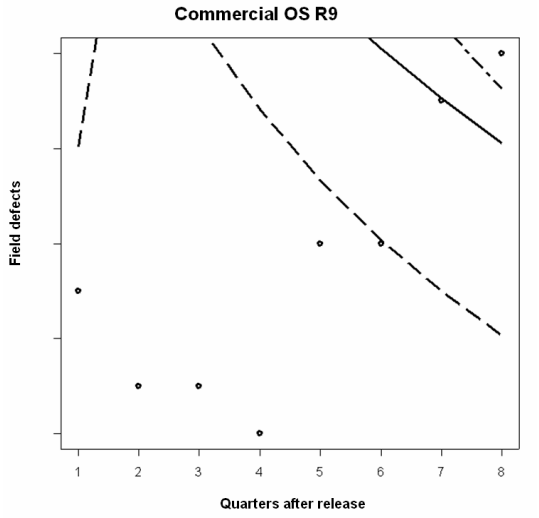

Figure 3. Actual and predicted field defects for OS Release 9

Since average Theil statistics are greater than or equal to 1 for all combinations, we conclude that is not possible to accurately forecast the field defect rate. This is also evident in the plots (figures 1-3). The predicted field defect occurrence patterns do not match the actual field defect occurrence patterns.

There are three contributing factors the may have contributed to the poor results: varying field defect occurrence patterns, lack of metrics, and long time intervals. We conjecture that these three factors are conditions under which the combined time-based and metrics-based approach will not yield accurate field defect rate forecasts.

5.1.1 Varying field defect occurrence patterns

The field defect occurrence patterns vary greatly between different releases. Figures 4-7 provides a sample of four releases and their best post-facto fits.

There are two implications. First, the diverse patterns mean that the Weibull model is best suited to model the field defect occurrence patterns since the Weibull is flexible enough to describe a wide range of patterns. This conjecture is supported by the fact that top two prediction methods

use the Weibull model. However, the model parameters of the Weibull are harder to predict. The errors in predictions are exaggerated by the Weibull model form (in section 4.1 table 2); therefore, forecasts are not accurate. This is the same conclusion reached by Li et al. in [15].

Secondly, since the field defect patterns vary greatly between releases, more data are needed to distinguish between releases; however, we have limited data (at most 6 training data points) in our study.

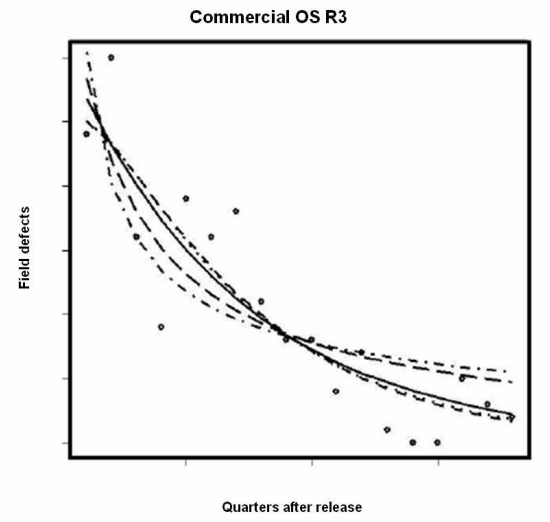

Figure 4. Actual field defects and fitted models for OS Release 3

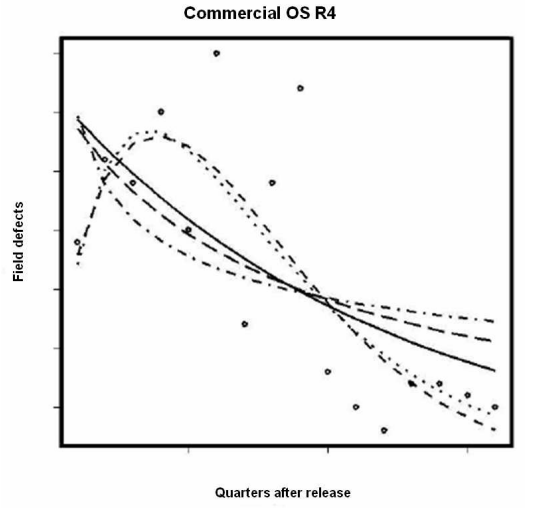

Figure 5. Actual field defects and fitted models for OS Release 4

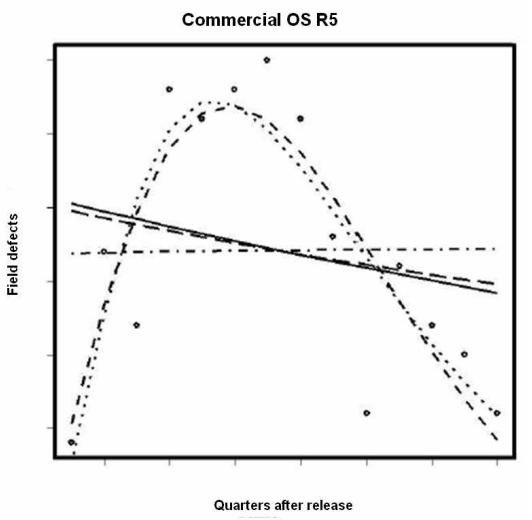

Figure 6. Actual field defects and fitted models for OS Release 5

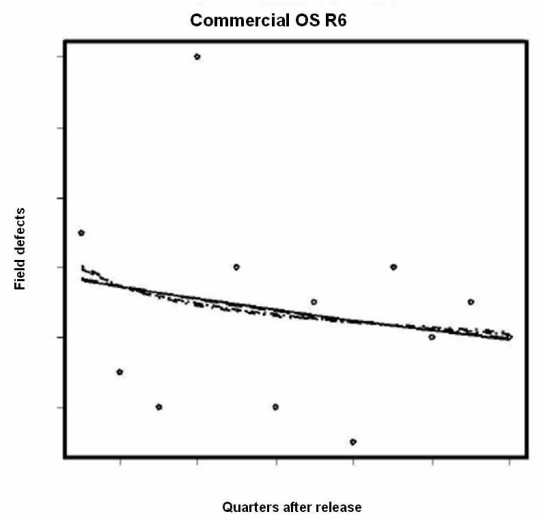

Figure 7. Actual field defects and fitted models for OS Release 6

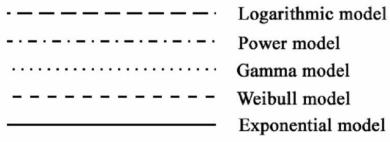

Figure 8. Legend for Figures 4-7

5.1.2 Lack of different categories of metrics

Li et al. [15] identifies each category of metrics (discussed in section 2.1) as important. The authors find that product, development, deployment and usage, and software and hardware configurations metrics are used for forecasts. However, in this study we only have 4 development metrics. In comparison, the referenced work had 22 development metrics and 145 metrics in all.

The differences in defect occurrence patterns may be due to varying deployment and usage characteristics as suggested by Li et al. in [15]; however, we are not able to collect deployment and usage information for this study.

The lack of metrics in different categories may have prevented the various modeling methods from building accurate models.

5.1.3 Long time intervals

The time interval between releases for operating system aggregated data is quarters (i.e. 3 months); however, in previous studies (e.g. Li et al. [15]) the time interval is months.

Since data is aggregated by quarters, for each additional release, only two additional data points become available for modeling. Models fitted with fewer points are more prone to be influenced by outliers and tend not to be stable (i.e. the fitted values are likely to change with additional data points). Since our forecasts are based on data available before release, the metrics-based models may have been trained with flawed data (i.e. poorly-fitted model parameters), which result in poor predictions.

5.2 The number of field defects predictions

Having determined that we are not able to accurately forecast the rate of field defects, we attempt to predict the number of field defects in the first six months after deployment. The top ten predictions ranked in terms of ARE are in table 4. The ratios method has the lowest ARE.

Table 4. Absolute relative error for field defects in the first 6 months

Method	*ARE*	*Std Error*
Ratios	0.9	1.2
Nonlinear Regression	1.0	0.7
Linear Regression	1.0	0.6
Neural Networks	1.1	0.8
Clustering	1.1	1.1
Reverse Smoothing 2 Releases	1.2	1.2
Moving Average 2 Releases	1.3	1.4
Exponential Smoothing 2 Releases	1.3	1.6
Trees Split with 2 Releases	1.4	1.2
Tree Split with 4 Releases	1.4	1.3

We conjecture that the stability of the development organization may have contributed to the results. Since the operating system is a mature product, the development process/effort is stable; therefore the software development organization may have been able to consistently remove the same percentage of problems for each release. Similar results in Khoshgoftaar et al. [16] suggest that simple approaches can produce accurate predictions for mature software development organizations. We feel the ratios method may allow other similar organizations to use the same approach to make initial predictions.

The only problem with using the ratios method is that the numbers of field defects found in the first six months are low. We find that on average the numbers of field defects in the first six months represent only 21% of the field defects found in the first two years.

6 CONCLUSION

In this paper, we report experiences from attempting to model field defects for an operating systems from a large software producing organization. We conjecture that it may not be possible to accurately forecast the rate of field defects if the field defect occurrence pattern varies greatly between releases, if predictors on product, development, deployment and usage, and software and hardware configurations are not available, and if the estimated field defect occurrence patterns are not stable due to insufficient data. We have also shown that the simple ratios method using development information can be used to predict the number of initial field defects.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grants ITR-0086003 and CCF-0438929, by the Carnegie Mellon Sloan Software Center, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298. This research is possible through an IBM joint study agreement.

REFERENCES

[1] Anderson, R.E. Social impacts of computing: Codes of professional ethics. Social Science Computing Review, 2 (Winter 1992), 453-469.

[2] Kathyrn Bassin and P. Santhanam. Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. In *Proceedings of ISSRE*, 1997.

[3] Shriram Biyani and P. Santhanam. Exploring Defect Data from Development and Customer Usage on Software Modules over Multiple Releases. In *Proceedings of ISSRE*, 1998.

[4] Michael Buckly and Ram Chillarege. Discovering Relationships between Service and Customer Satisfaction. In *Proceedings of the International Conference on Software Maintenance* 1995.

[5] Norman Fenton and Martin Neil. Software metrics: road map. In *Proceedings of ICSE*, 2000.

[6] Donald E. Harter and Mayuram S. Krishnan and Sandra A. Slaughter. Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. In *Management Science*, 2000.

[7] Wendell Jones, John Hudepohl, Taghi Khoshgoftaar, and Edward Allen. Applications of a Usage Profile in Software Quality Models. In *3rd European Conference on Software Maintenance and Reengineering*, 1999.

[8] Garrison Kenny. Estimating Defects in Commercial Software during Operational Use. In *IEEE Tr. on Reliability*, 1993.

[9] Taghi M. Khoshgoftaar and Edward B. Allen. Predicting fault-prone software modules in embedded systems with classification trees. In *IEEE Symposium on High-Assurance Systems Engineering*, 1999.

[10] Taghi Khoshgoftaar, Bibhuti Bhattacharyya, and Gary Richardson. Predicting Software Errors, During Development, Using Nonlinear Regression Models: A Comparative Study. In *IEEE Tr. On Reliability*, 1992.

[11] Taghi M. Khoshgoftaar and Edward B. Allen and Kalai S. Kalaichelvan and Nishith Goel. Early Quality Prediction: A Case Study in Telecommunications. In *IEEE Software*, 1996.

[12] Taghi Khoshgoftaar, John Munson, and David Lanning. A Comparative Study of Predictive Models for Program Changes during System Testing and Maintenance. In *Proceedings of ICSM*, 1993.

[13] Taghi Khoshgoftaar, Abhijit Pandya, and David Lanning. Application of Neural Networks for Predicting Program Fault. In *Annals of Software Engineering*, 1995.

[14] Taghi Khoshgoftaar and Naeem Seliya. Tree-based Software Quality Estimation Models for Fault Prediction. In *Proceedings of METRICS*, 2002.

[15] Taghi Khoshgoftaar and Naeem Seliya. Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques. In *Empirical Software Engineering*, vol 8, Sep 2003, p255-283.

[16] Taghi Khoshgoftaar and Naeem Seliya. Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study. In *Empirical Software Engineering Journal*, vol 9, Sep 2004, p229-257.

[17] Paul Luo Li and Jim Herbsleb and Mary Shaw. Forecasting Field Defect Rates Using a Combined Time-based and Metrics-based Approach: a Case Study of OpenBSD. Submitted to *ISSRE*, 2005.

[18] Paul Luo Li and Mary Shaw and Jim Herbsleb and Bonnie Ray and P.Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. In *Proceedings of FSE,* 2004.

[19] Paul Luo Li, Mary Shaw, Kevin Stolarick, and Kurt Wallnau. The Potential for synergy between certification and insurance. In *Special edition of ACM SIGSOFT Int'l Workshop on Reuse Economics (in conjunction with ICSR-7),* 2002.

[20] Michael Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[21] Audris Mockus, David Weiss, and Ping Zhang. Understanding and Predicting Effort in Software Projects. In *Proceedings of ICSE,* 2003.

[22] Audris Mockus and Ping Zhang and Paul Luo Li. Drivers for Customer Perceived Quality. In *Proceedings of ICSE*, 2005.

[23] Nachiappan Nagappan and Thomas Ball. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of ICSE,* 2005.

[24] National Institute of Standards and Technology. *The economic impacts of inadequate infrastructure for software testing.* Planning Report 02-3, 2002

[25] Thomas Ostrand, Elaine Weyuker, and Thomas Bell. Where the Bugs are. In *Proceedings of ISSTA*, 2004.

[26] The R project for statistical computing. www.r-project.org

[27] Norman F. Schneidewind. Body of Knowledge for Software Quality Measurement. In *IEEE Computer*, 2002

[28] Henri Theil. *Applied Economic Forecasting.* North-Holland Publishing Company Netherlands, 1966.