Forecasting Field Defect Rates Using a Combined Time-based and Metrics-based Approach: a Case Study of OpenBSD

Paul Luo Li Jim Herbsleb Mary Shaw
Institute for Software Research, International,
School of Computer Science,
Carnegie Mellon University
Pittsburgh PA, 15213
1-412-268-3043
{paul.li, jdh, mary.shaw} @cs.cmu.edu

ABSTRACT

Open source software systems are critical infrastructure for many applications; however, little has been precisely measured about their quality. Forecasting the field defect-occurrence rate over the entire lifespan of a release before deployment for open source software systems may enable informed decision-making. In this paper, we present an empirical case study of ten releases of OpenBSD. We use the novel approach of predicting model parameters of software reliability growth models (SRGMs) using metrics-based modeling methods. We consider three SRGMs, seven metrics-based prediction methods, and two different sets of predictors. Our results show that accurate field defect-occurrence rate forecasts are possible for OpenBSD, as measured by the Theil forecasting statistic. We identify the SRGM that produces the most accurate forecasts and subjectively determine the preferred metrics-based prediction method and set of predictors. Our findings are steps towards managing the risks associated with field defects.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – Reliability, Statistical methods
D.2.8 [**Software Engineering**]: Metrics – Process metrics, Product metrics, Software science
D.2.9 [**Software Engineering**]: Management – Cost estimation, Software quality assurance

General Terms

Management, Measurement, Reliability, Experimentation

Keywords

Metrics-based modeling, deployment and usage metrics, software and hardware configurations metrics, comparative study, open source software

1. INTRODUCTION

Many software applications, including mobile applications, depend upon open source software systems to provide critical computing infrastructure. The quality of the infrastructure (e.g. operating system) may affect the quality of the application. In this paper, we present a case study of the open source operating system OpenBSD, which is a key component of several commercial network security products [30].

Quantitatively-based decision making regarding open source systems is often difficult, because the quality of open source software systems is often not known quantitatively. Being able to forecast field defect-occurrence rates (i.e. the rates of customer reported software problems requiring developer intervention to resolve) over the entire lifespan of a release (i.e. as long as there are field defect occurrences) before deployment (i.e. at the time of release) may allow existing quantitatively-based decision-making methods to be used to:

- Help organizations seeking to adopt open source software systems to make informed choices between candidates
- Help organizations using open source software systems to decide whether to adopt the latest release
- Help organizations that adopt a release to better manage resources to deal with possible defects
- Insure users against the costs of field defect occurrences

Prior work by Li et al. [17] has shown that it is not possible to forecast field defect-occurrence rates (i.e. the field defect-occurrence pattern over time) by fitting a SRGM to development defect information. In this paper, we report results using the novel approach of using metrics-based modeling methods to predict model parameters of time-based models (i.e. SRGMs).

We conduct empirical experiments comparing combinations of SRGMs, metrics-based modeling methods, and sets of predictors to forecast field defect-occurrence rates before release. We construct combinations along the following dimensions:

Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05)
1071-9458/05 $20.00 © 2005 **IEEE**

1) Type of SRGM: Which SRGM yields the most accurate field defect-occurrence rate forecasts?
   a. Weibull model, described in Kenny [4]
   b. Gamma model, described in Lyu [20]
   c. Exponential model, described in Musa et al. [24]
2) Modeling methods: Which metrics-based modeling method predicts model parameters that produce the most accurate field defect-occurrence rate forecasts?
   a. Moving averages, used in Li et al. [15]
   b. Exponential smoothing, used in Li et al. [15]
   c. Linear regression with model selection, used in Khoshgoftaar et al. [11] and Khoshgoftaar et al. [8]
   d. Principal component analysis, clustering, and linear regression, used in Khoshgoftaar et al. [10]
   e. Trees, using used in Khoshgoftaar and Seliya [13]
   f. Non-linear regression, used in Khoshgoftaar and Munson [9] and Khoshgoftaar et al. [8]
   g. Neural networks, used in Khoshgoftaar et al. [12] and Khoshgoftaar et al. [11]
3) Predictors: Do more predictors and more categories of predictors yield more accurate forecasts?
   a. The same kinds of predictors as the referenced work (e.g. product metrics only)
   b. A superset of predictors that includes 145 predictors (product metrics, development metrics, deployment and usage metrics, and software and hardware configurations metrics)

We empirically compare the accuracy of forecasts for nine releases of OpenBSD. We use the Theil forecasting statistic to measure the accuracy of forecasts. Theil statistics lower than 1 are considered accurate (discussed in section 5). We subjectively determine the best model, modeling method, and set of predictors by considering the accuracy of predictions and the amount of information needed before a prediction can be made

Our results show that the simple Exponential model produces more accurate forecasts (i.e. forecasts with lower Theil statistics) than the more complex Gamma and Weibull models. The trees method is the best metrics-based prediction method because it predicts model parameters that yield forecasts ranked in the top 10 in terms of accuracy and because the trees method is able to make predictions with limited data. Our results show that it is possible to make predictions ranked in the top 10 in terms of accuracy without using the superset of predictors.

Theil statistics of our forecasts indicate that our approach yields accurate forecasts. Our results enable future work to examine the adequacy of forecasts for quantitatively-based decision making methods.

We present prior work, which serves as motivation for our work, in section 2. We describe OpenBSD in section 3. Our data collection and data analysis techniques are discussed in sections 4 and 5. Section 6 presents our results. We present a discussion in section 7 and conclude in section 8.

2. PRIOR WORK AND EXPERIMENTAL DESIGN

We motivate our work and our experimental design by discussing prior work.

We define a *field defect* as a user-reported problem occurring after release requiring developer intervention to resolve. Our operational measure of a field defect for OpenBSD is a user submitted problem report in the request tracking system of the class software bugs occurring after the official release date (discussed more in sections 3 and 4). Each problem report is counted. For example, two user-reported problems traced to the same underlying defect are counted as two field defects. These software related problem reports require developer intervention to resolve. A *field defect occurrence* is the occurrence of a field defect. A similar definition is used in Li et al. [15].

2.1 Fixed dimensions in our experimental design

Granularity of observation, types of prediction, defect modeling approaches, and forecasting approaches are dimensions of variation we do not vary in our study. The dimensions listed in the introduction are dimensions we vary in out study and are discussed in section 2.2.

2.1.1 Granularity of observation

In this paper, we examine field defect occurrences for the entire system as a whole. This is the correct level of granularity because we are focused on helping software consumers; and, software consumers generally view the software system as a whole.

Prior work has predicted field defects for individual software changes (e.g. in Mockus et al. [21]), files (e.g. in Ostrand et al. [26]), modules (e.g. in Khoshgoftaar et al. [12]), and entire systems (e.g. in Kenney [4]).

2.1.2 Types of predictions

In this paper, we predict the rate of field defect occurrences over time because effective quantitatively-based decision making requires knowing the rate of field defect occurrences over time as discussed by Li et al. [15].

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

- Relationships: These studies establish relationships between predictors and field defects. For example, Harter et al. [2] establish a relationship between an organization's CMM level and the number of field defects.
- Classifications: These studies predict if the number of field defects is above a threshold for an observation. For example, Khoshgoftaar et al. [6] classify software modules as risky (will contain at least one field defect) or not risky (no field defects).

- Quantities: These studies predict the number of field defects. For example, Khoshgoftaar et al. [11] predict the number of defects for software modules.
- Rates of occurrences over time: These studies predict the field defect-occurrence rate. For example, Kenny [4] predicts the defect occurrence pattern as captured by the Weibull model for two IBM systems.

2.1.3 Defect modeling approaches

In this paper, we use a novel approach of using metrics-based modeling methods to predict model parameters of a SRGM, which captures the field defect-occurrence pattern of a software release over the entire lifetime of the release (i.e. until there are no more field defect occurrences).

Field defect predictions generally belong to one of two classes: time-based approach and metrics-based approach. Schneidewind [28] distinguishes 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 SRGM. The field defect–occurrence rate is forecasted using the fitted SRGM. Musa [20] and Lyu [24] describe 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 predict classifications or quantities; however, metrics-based models have not been used to predict model parameters of SRGMs. Examples of this approach are in Mockus [22] and Khoshgoftaar et al. [11]

Li et al. [17] show that it is not possible to use the time-based approach of fitting a SRGM to development defects to predict field defect-occurrence rates for OpenBSD. The authors find that the field defect-occurrence rates are generally increasing at the time of release; therefore, the authors cannot fit a meaningful model. Other studies (e.g. [16] and [4]) reach similar conclusions.

Furthermore, 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 (as stated 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 for these systems. Therefore, it may not be appropriate to forecast field defect-occurrence rates using a SRGM fitted using testing information.

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

2.1.4 Forecasting approaches

In this paper, we simulate a real world situation by forecasting field defect-occurrence rates using only information available at the time of release (i.e. before deployment) for multiple releases.

Prior work in metrics-based modeling either inadequately addresses multiple releases or does not account for multiple active releases. Some studies (e.g. Khoshgoftaar et al. [11]) 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. A better approach is to use a model fitted using data from a historical release to predict for future releases. This is the approach taken by Khoshgoftaar et al. in [6] and by Ostrand et al in [26]. However, previous studies assume that complete 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.

In this study, we estimate model parameters for active historical releases using field defect information available at the time of release. An example prediction situation for a typical release is illustrated in Figure 1.

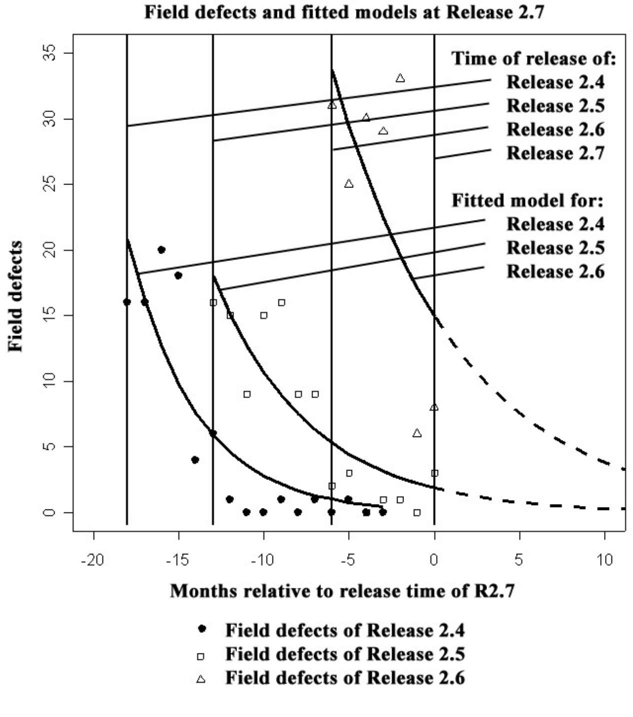

Figure 1. Example fitting situation

At the time of release of release 2.7, predictor information is available for releases 2.4-2.7 and complete field defect information (i.e. model parameters of the fitted model) is available for release 2.4. However, releases 2.5 and 2.6 are still active (i.e. field defects are still occurring); therefore, we use the estimated model parameters for the two releases.

Predictor information and model parameters for releases 2.4-2.6 are then used to predict model parameters for release 2.7.

2.2 Dimensions of variation in our experimental design

The SRGMs, the modeling methods, and the predictors are the dimensions we vary in our study.

2.2.1 Software reliability growth models (SRGMs)

Prior work by Li et al. [15] has compared the ability of SRGMs from the literature to model the rate of defect occurrences (including defects during development) of OpenBSD based on post-facto fits and has concluded that the Weibull model is better than other models, as judged by the AIC model selection criterion. We have replicated the experiment using only field defects and have arrived at the same conclusions (i.e. the Weibull model is better) [18].

Prior work is based on post-facto fits evaluated using the AIC model selection criterion [15]. Even though AIC penalizes for extra model parameters, Weibull model parameters may be much harder to predict compared with model parameters of other models. Therefore, in this paper, we also consider the Gamma model (also known as the S-shaped model [20]) and the Exponential model (also known as the Goel-Okumoto model [20]), which have been shown to be the next most effective models [18]. We have also examined the Logarithmic (also known as the Musa-Okumoto model [24]) and Power (also known as Duane's model [20]) models; however, their post-facto fits are worse than the models we consider for releases of OpenBSD.

The models' forms are in table 1. The model parameters (highlighted) dictate the rate of field defect occurrences. We predict the model parameters using metrics-based modeling methods. Interpretations of the models and discussions of the match between the SRGMs and the field defect-occurrence phenomenon (e.g. in Musa [24] and in Kenny [4]) are beyond the scope of this paper. This dimension of variation addresses the question:

Which SRGM yields the most accurate field defect-occurrence rate forecasts?

Table 1. Software reliability models

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

2.2.2 Metrics-based modeling methods

Prior work has explored using metrics-based modeling methods to predict quantities (e.g. the total number of field defects). It may be possible to use these methods to predict model parameters that describe the field defect-occurrence pattern. We consider metrics-based modeling methods that have been used in previous studies to predict quantities. We discuss these methods in detail in section 5.

Many studies have compared the accuracy of predicted classifications of various metrics-based models (e.g. Khoshgoftaar et al. [7]). Few studies have compared the accuracy of predicted quantities of various metrics-based models (e.g. Khoshgoftaar et al. [11]). No work has compared the accuracy of predicted field defect-occurrence rates of various metrics-based methods. This dimension of variation addresses the question:

Which metrics-based modeling method predicts model parameters that produce the most accurate field defect-occurrence rate forecasts?

2.2.3 Predictors

Metrics available before release are *predictors,* which can be used by metrics-based modeling methods to predict model parameters.

We categorize predictors used in prior work using an augmented version of the categorization schemes used by Fenton and Pfleeger in [1] and by Khoshgoftaar and Allen in [5]:

- Product metrics: metrics that measure 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. [6].
- 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. [3].
- 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].

Prior work has only examined commercial software systems, and no prior work has examined predictions using predictors in all the categories simultaneously. In this paper, we compare using only predictors in the referenced work (e.g. product metrics only) and using a superset of predictors (i.e. predictors in all the categories). This dimension of variation addresses the question:

Do more predictors and more categories of predictors yield more accurate forecasts?

3. SYSTEM DESCRIPTION

OpenBSD is an open source Unix-style operating system written primarily in C. The OpenBSD project uses the Berkley copyrights. The Berkley copyrights retain the rights of the copyright holder, while imposing minimal conditions on the use of the copyrighted material; therefore, OpenBSD has been incorporated into several commercial products.

The OpenBSD project puts out a release approximately every six months. The release dates are published on the web. The OpenBSD project manages its source code using a CVS code repository, uses a problem tracking system, has multiple mailing lists. The project dates back to 1995 and is described in more detail in Li et al. [17].

We examined the project between approximately 1998 and 2004. During that time, there were 11 releases (of which we examine 10, as we explain below).

4. DATA COLLECTION

We consider the published date of release (announced on the OpenBSD website) rounded to the nearest month to be the release date for the release. We round the release date to the next month (i.e. a ceiling function) due to the time it takes to install the system, use the system, discover a problem, and the report the problem. Someone reporting a bug right after the un-rounded release date is unlikely to be using the latest release. Mockus et al. use the same approach in [22].

4.1 Data extraction

We briefly discuss our data extraction process. A detailed description is in Li et al. [17].

We wrote Java and perl programs to download problem reports from the OpenBSD website and to extract the number of messages in the mailing lists archives.

There was one anomaly. Three months of field defect-occurrence data were missing between August 2002 and October 2002. We verified this by examining the bugs mailing list archive (i.e. the mailing lists that records messages to the request tracking system). The mailing list archive showed no bugs recorded during that time interval even though there was activity on the bugs mailing list, which indicated that problems did occur. This happened during development and deployment of release 3.2. As a result, we did not examine release 3.2.

We used the CVS checkout command to download the source code from the CVS repository for releases 2.4 to 3.4 (except release 3.2). We then used five metrics tools and several scripts to compute product metrics for the C source files. We computed predictors for each file then summed the predictors for all files in the system.

4.2 Predictor computation

We briefly discuss the predictors we collect. A detailed description of the predictors is in Li et al. [17].

We attempted to collect the same metrics as the referenced studies (discussed in section 5). We collected the same metrics when possible and collected metrics that capture the same intent otherwise. All the predictors used in previous studies were product metrics. We computed product metrics (106 metrics) and development metrics (22 metrics) that capture each sources of variance in product and development metrics identified by Munson and Khoshgoftaar in [23] and by Khoshgoftaar et al. in [14]. Furthermore, we computed metrics that capture information about deployment and usage (9 metrics) and software and hardware configurations in use (7 metrics).

We collected deployment and usage metrics in two categories: mailing list predictors and request tracking system predictors. Mailing list predictors counted the number of messages to non-hardware related mailing lists during development. We believed our mailing list predictors were valid because they quantified the amount of interest in OpenBSD, which might be related to deployment and usage. Request tracking predictors counted the number of problem reports during development that were not defects (e.g. documentation problems). We believed our request tracking system predictors were valid because users had to install OpenBSD and use the system before they could report a problem. An example of a deployment and usage metric is *TechMailing,* which is the number of messages to the technical mailing list during the development period.

We collected software and hardware configuration metrics in two categories: mailing list predictors and request tracking system predictors. Mailing list predictors counted the number of messages to hardware specific mailing lists during development. We believed our mailing list predictors were valid because they reflected the amount of interest/activity related to the specific hardware, which might be related to how many of the specified hardware machines had OpenBSD installed. Request tracking predictors counted the number of defects (field defects and development defects) during development that identified the type of hardware used. We believed our request tracking system predictors were valid because users had to install OpenBSD on the specified HW before they could report a problem. An example of a software and hardware configurations metric is *AllDefectHWSparc,* which is the number of field defects reported against all active release during the development period that identify the machine as of type Sparc.

5. DATA ANALYSIS

In this section, we describe the modeling methods in each referenced work as well as the adjustments we had to make. A more detailed discussion is in [18].

We predicted model parameters using each of the metrics-based modeling method (the same method for all model parameters). Accuracy of the resulting field defect-occurrence rate forecast was evaluated using the Theil forecasting statistic. Analysis was preformed using the open source statistical package R [27].

The Theil statistic compares the forecast for each time interval *i* against a no-change forecast based on the previous time interval's value [29].

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

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.

5.1 Principal component analysis, clustering, and linear regression

We roughly replicated (explained below) the principal component analysis (PCA), clustering, and linear regression method in Khoshgoftaar et al. [10]. PCA constructs new predictors that capture all the variation in the original predictors using linear combinations of the original predictors. Clustering groups observations together based on predictors' values.

Khoshgoftaar et al. [10] constructed principal components and then clustered observations using the principal components. They fitted linear models 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 for the new observation.

Khoshgoftaar et al. [10] predicted field defects for modules using 11 product metrics. They fitted models using 260 observations in four clusters. Since we only had 9 observations, we modified the process to use two clusters and to fit a null linear model for each cluster (i.e. an average of the observations). In addition, we did not have enough observations to perform a PCA. Therefore, when using the same predictors as the original study, we used the linear coefficients of the referenced work to construct principal components. When using all the predictors, we did not conduct a PCA. We used the popular K-means clustering method, since Khoshgoftaar et al. [10] did not identify the clustering method used.

5.2 Linear regression with model selection

We replicated the linear regression with model selection method in Khoshgoftaar et al. [11] and in Khoshgoftaar et al. [8]. Linear regression models the predicted value using a linear combination of predictors' values. Model selection keeps predictors that improve the fit significantly as judged by a model selection criterion (e.g. AIC).

Khoshgoftaar et al. [11] and Khoshgoftaar et al. [8] used backwards and stepwise model selection techniques to select a subset of predictors. They fitted a linear regression model 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. [11] and Khoshgoftaar et al. [8] predicted field defects for modules of two systems using 8 product metrics for one system and 11 product metrics for the other system. They 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 Khoshgoftaar et al. [11] and Khoshgoftaar et al. [8], we used the popular AIC model selection criterion.

5.3 Non-linear regression

We replicated the non-linear regression method used in Khoshgoftaar and Munson [9] and in Khoshgoftaar et al. [8]. Non-linear regression models the predicted value using a non-linear combinations of the predictors' values.

Khoshgoftaar and Munson [9] and Khoshgoftaar et al. [8] used non-linear least squares regression to construct non-linear models 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 predictor was inserted into the fitted model to produce a prediction.

Khoshgoftaar and Munson [9] and Khoshgoftaar et al. [8] used 15 observations to train the model. We found that it was not possible to fit a model with three parameters using 9 observations; therefore, we simplified the model by dropping a modeling parameter. Our model was:

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

5.4 Trees

We replicated the Classification and Regressions Trees (CART) method in Khoshgoftaar and Seliya [13]. The trees method iteratively splits observations into similar groups as judged by the predicted value using predictors' values.

Khoshgoftaar and Seliya [13] built a regression tree using training observations and a minimum node size of 10. To predict for a new observation, the observation traversed the tree until it reached a leaf node. The mean of the training observations in the leaf node was the predicted value of the new observation.

Khoshgoftaar and Seliya [13] predicted field defects in modules using 9 product metrics. They fitted models using 4648 observations. Since we had at most 9 training observations, we built trees with varying minimum node sizes of between 2 to 7.

5.5 Neural networks

We replicated the feed-forward neural networks method used in Khoshgoftaar et al. [12] and Khoshgoftaar et al. [11]. Neural networks use non-linear functions to combine predictors' values to produce an output.

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. A non-linear function is used to combine values to connect layers and to produce the output. 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. [12] and Khoshgoftaar et al. [11] 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. 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. [12] and Khoshgoftaar et al. [11] predicted field defects for the same two systems as the linear regression with model selection method. They 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 best model by evaluating fitted values. The most accurate model was then used to make predictions for the next release.

5.6 Exponential smoothing and moving averages

We replicated the moving averages and exponential smoothing methods used in Li et al. [15]. 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, since recent releases might be more similar to the current release. Li et al. [15] considered averaging 2-7 releases. We made no modifications to the method.

6. RESULTS

This section summarizes results of our 99 forecasting experiments. The top 10 SRGM, prediction method, and predictors combinations based on the average Theil statistic are in table 2. Complete results are in [18].

No training data was available for the first release (R2.4) and we excluded release 3.2; therefore, we predicted for nine releases. Many combinations were not able to predict for all releases because the modeling methods required additional data.

Our approach yields accurate forecasts, as measured by the Theil statistic (discussed in section 5). The accuracy is also evident upon a visual inspection of our forecasts. A plot of the nine releases and forecasts of the top three combinations are in figure 2.

Table 2. Theil forecasting statistics

Model, method, predictor combination	*R2.5*	*R2.6*	*R2.7*	*R2.8*	*R2.9*	*R3.0*	*R3.1*	*R3.3*	*R3.4*	*Avg*
Exponential model using the moving averages method of 2 releases using no predictors		.7520	.5911	.5267	.3099	.5982	.6925	.6142	.4360	.5651
Exponential model using the non-linear regression method using lines of code (same predictors as referenced work)				.7017	.3172	.7830	.6788	.4023	.5079	.5651
Exponential model using the trees method splitting with six observations using all predictors	**.7048**	**.7520**	**.4407**	**.6978**	**.2984**	**.5713**	**.6745**	**.6754**	**.2991**	**.5682**
Exponential model using the exponential smoothing method of five releases using no predictors				.2973	.6795	.6795	.6858	.6058	.6547	.5846
Gamma model using the non-linear method using lines of code (same predictors as referenced work)				.6690	.4052	.7056	.6590	.4393	.6412	.5866
Exponential model using the exponential smoothing method of four releases using no predictors			.6462	.3222	.3222	.6469	.6890	.6117	.6180	.5890
Exponential model using the moving averages method of four releases using no predictors			.6978	.3047	.3047	.6418	.6883	.5264	.6854	.5907
Exponential model using the exponential smoothing method of two releases using no predictors		.6436	.6436	.5365	.3577	.6202	.6926	.6746	.4386	.5908
Exponential model using trees method splitting on with 7 releases using all predictors	.7048	.7520	.4407	.6978	.2983	.7854	.6745	.6754	.2991	.5920
Exponential model using the moving averages method of three releases using no predictors		.4407	.6504	.6166	.3695	.6610	.6926	.6834	.6207	.5932

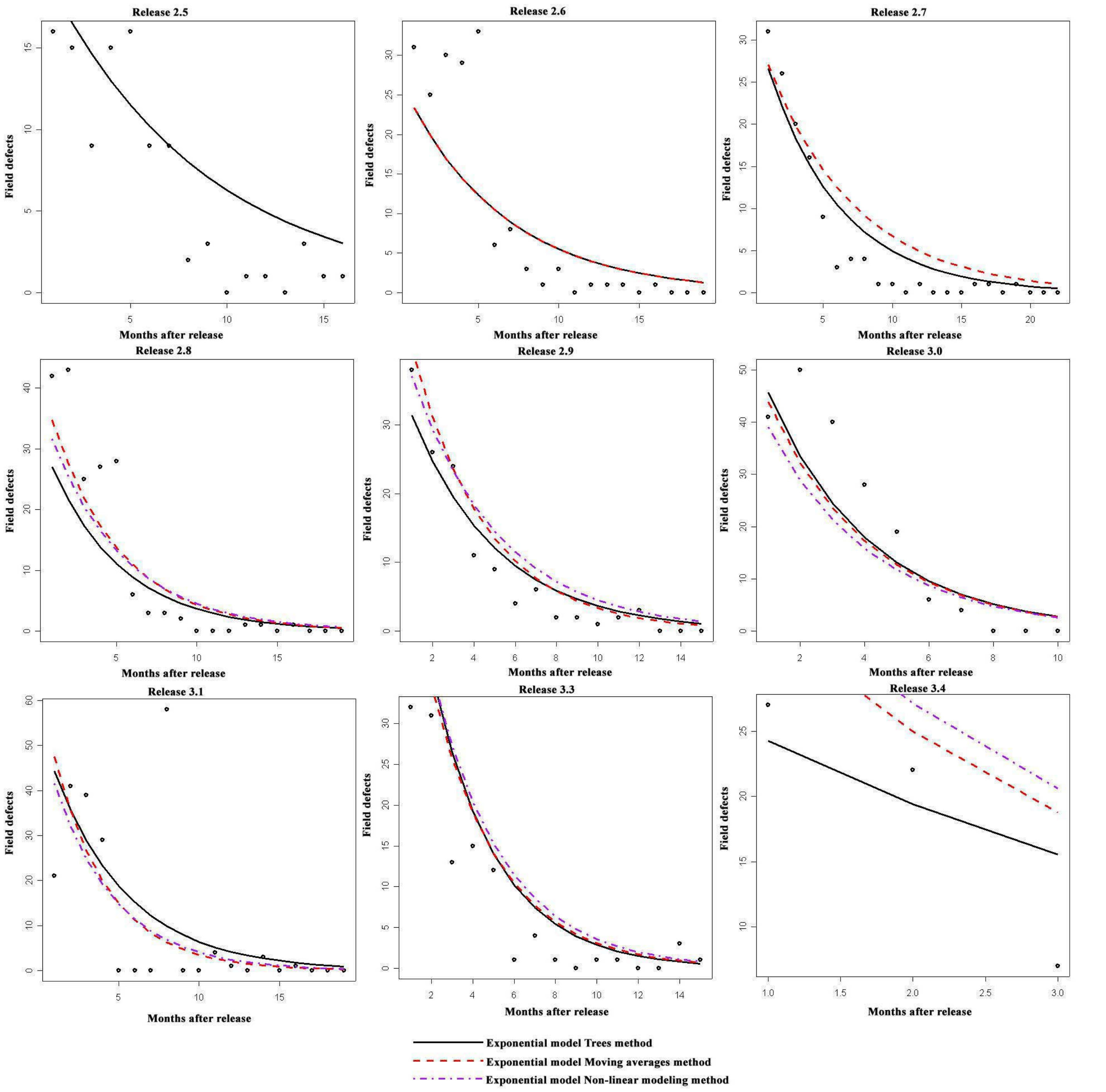

Figure 2. Predicted defect-occurrence rates at the time of release

The trees method splitting with a minimum of six observations using the Exponential model and all predictors is the best combination (highlighted in table 2). It is able to predict for all releases and its average Theil statistic is within .0032 of the best Theil statistic. In addition, of the top ten combinations, it has the best Theil statistics for 6 out of the 9 releases (more than any other combination) and its Theil statistics is within .401 of the best Theil statistics for all releases. The predictors used in the trees are in table 3. The fitted trees for the two parameters of the Exponential model for Release 3.4 (the most recent release) are in figures 3 and 4.

Table 3. Predictors used

Metric	*Definition*	*Prediction used*
AllDefectHW Sparc	Field defects reported during the development period that identify the machine as of type Sparc	parameter N for R3.0 and R3.3
LOC	Lines of code	parameter α for R3.0 and N for R3.1
Comment Inline	Inline comments	parameter α for R3.1 and R3.3
TechMailing	Messages to the technical mailing list during the development period	parameter N for R3.4
NotCUpdate	Updates (deltas) to non-c-source-files during the development period	parameter α for R3.4

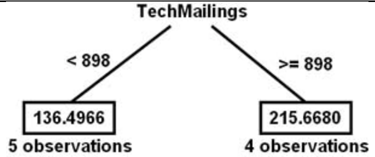

Figure3. Fitted CART for Exponential parameter N for release 3.4

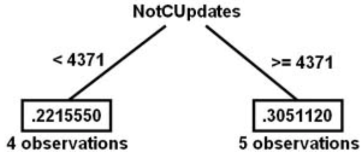

Figure 4. Fitted CART for Exponential parameter α for release 3.4

7. DISCUSSION

In this section, we present our conclusions regarding SRGMs, modeling methods, and predictors based upon our results.

7.1 Reliability models

Our results indicate that the simple Exponential model is better than more complex models like Gamma and Weibull models when forecasting field defect-occurrence rates before deployment.

Post facto fits had shown the Weibull model to be the best model based on AIC, which penalized for extra model parameters. However, in our experiments, nine out of the top ten combinations used the Exponential model. The Exponential model had only two model parameters that needed to be predicted. The Weibull and Gamma models each had three. In addition, the model form of the exponential model was simpler. The Exponential model did not have a power term, thus errors in parameter predictions were not magnified. These factors might have contributed to better forecasts using the Exponential model.

7.2 Modeling methods

Our results indicate that the trees method can predict model parameters that result in accurate forecasts even when data are scarce.

We had at most 9 training observations (in a real world setting, more data is unlikely to be available). Other metrics-based modeling techniques might not have been effective because they did not have enough training data. For example, the neural network method in Khoshgoftaar et al. [12] and Khoshgoftaar et al. [11] had ~20x more training observations. If more data were available, other metrics-based methods might have produced better results. However, the trees method was effective even though Khoshgoftaar and Seliya [13] had ~500x more training observations. This supported our conclusion that the trees method was the best method.

7.3 Predictors

Our results indicate that accurate forecasts (i.e. forecasts that are in the top ten in terms of the Theil forecasting statistic) are possible even with few (e.g. only lines of code) or no predictors.

Six out of ten combinations in the top ten were moving averages or exponential smoothing methods. They did not use any predictors. Of the other four methods in the top ten, two used all the predictors (trees methods) and two used only lines of code (non-linear regression methods).

First, since we collected 145 predictors and had at most 9 observations in the training set, spurious fits (i.e. fits that are better by chance) might have occurred. This might have reduced the benefits of having more predictors.

When all the predictors were used, the important predictors included predictors capturing characteristics of the development process (NotCUpdates), of the deployment and usage pattern (TechMailings), and of the software and hardware configurations in use (AllDefectHWSparc). Out findings supported previous findings that non-product related metrics are important predictors of field defects (e.g. Mockus et al. [22]).

Secondly, as evident in figure 2, the field defect-occurrence patterns of OpenBSD releases were very similar and thus changes in predictors did not correspond to changes in model parameter values. The developers of OpenBSD might have been able to evaluate their ability to implement features and to fix defects. Thus, the releases were released with similar quality and similar field defect occurrence patterns. The field defect-occurrences rates peaked within 3 months of the release date for all but two of the releases,.

8. CONCLUSION

In this case study, we have forecasted field defect occurrence rates over the entire lifespan of releases using only information available before release for OpenBSD using a novel approach of combining the time-based approach and the metrics-based approach. The results are interesting and appropriate for a case study; however, they need to be replicated to show general applicability. We envision replicating our experiment for commercial systems to examine differences due to development methods, as well as for other open source software systems.

We have shown that accurate forecasts are possible, as measured by the Theil forecasting statistic; however we have not determined if the forecasts are accurate enough for quantitatively-based decision making methods. Future work needs to address the issue. Confidence bounds and intervals also need to be considered.

We have tried to replicate modeling methods and to collect the same metrics as in previous studies. However, there may be differences due to specific definitions and modeling tuning parameters. These differences are acceptable for empirical replications as discussed by Ohlsson and Runeson in [25].

Our field defect-occurrence rates forecasts are steps towards quantitatively-based decision making, which can lower the risks associated with field defect occurrences.

9. 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. We would like to thank the developers of OpenBSD for their insight and tool vendors who gave us trial licenses.

10. REFERENCES

[1] Norman Fenton and Martin Neil. Software metrics: road map. *Proc. ICSE*, May 2000, pp. 357-370.

[2] 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. *Management Science,* vol. 46 no. 4, Apr 2000, pp. 451-466.

[3] Wendell Jones, John Hudepohl, Taghi Khoshgoftaar, and Edward Allen. Applications of a Usage Profile in Software Quality Models. *Proc. 3rd European Conference on Software Maintenance and Reengineering*, Mar 1999, pp. 148-157.

[4] Garrison Kenny. Estimating Defects in Commercial Software during Operational Use. *IEEE Tr. on Reliability,* vol. 42 no. 1, Mar 1993, pp. 107-115.

[5] Taghi M. Khoshgoftaar and Edward B. Allen. Predicting fault-prone software modules in embedded systems with classification trees. *Proc. HASE*, Nov 1999, pp. 105-112.

[6] Taghi Khoshgoftaar, Edward Allen, and Jianyu Deng. Controlling Over-fitting in Software Quality Models: Experiments with Regression Trees and Classification. *Proc. METRICS,* Apr 2001, pp. 190-198.

[7] Taghi M. Khoshgoftaar and Edward B. Allen and John P. Hudepohl and Stephen J. Aud. Application of Neural Networks to Software Quality Modeling of a Very Large Telecommunications System. *IEEE Tr. on Neural Networks*, vol. 8 no. 4, Jul 1997, pp. 902-909.

[8] Taghi Khoshgoftaar, Bibhuti Bhattacharyya, and Gary Richardson. Predicting Software Errors, During Development, Using Nonlinear Regression Models: A Comparative Study. *IEEE Tr. On Reliability,* vol. 41 no. 3, Sep 1992, pp. 390-395.

[9] Taghi Khoshgoftaar and John Munson. Predicting Software Development Errors using Software Complexity Metrics. *IEEE J. Selected Areas in Communications,* vol. 8 no. 2, Feb 1990, pp. 253-261.

[10] Taghi Khoshgoftaar, John Munson, and David Lanning. A Comparative Study of Predictive Models for Program Changes during System Testing and Maintenance. *Proc. ICSM,* Sep 1993, pp. 72-79.

[11] Taghi Khoshgoftaar, Abhijit Pandya, and David Lanning. Application of Neural Networks for Predicting Program Fault. *Annals of Software Engineering,* vol. 1, 1995, pp. 141-154.

[12] Taghi Khoshgoftaar, Abhijit Pandya, and Hamant More. A Neural Networks Approach for Predicting Software Development Faults. *Proc. ISSRE,* Oct 1992, pp. 83-89.

[13] Taghi Khoshgoftaar and Naeem Seliya. Tree-based Software Quality Estimation Models for Fault Prediction. *Proc. METRICS,* Jun 2002, pp. 203-214.

[14] Taghi Khoshgoftaar, Vishal Thaker, and Edward Allen. Modeling Fault-prone Modules of Subsystems. *Proc. ISSRE*, Oct 2000, pp. 259-267.

[15] Paul Luo Li, Mary Shaw, Jim Herbsleb, Bonnie Ray, and P. Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. *Proc. FSE,* vol. 29 no. 6, Oct 2004, pp.263-272.

[16] Paul Luo Li, Mary Shaw, Jim Herbsleb, Bonnie Ray, and P. Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. *CMU Tech Report CMU-ISRI-04-130,* 2004

[17] Paul Luo Li, Jim Herbsleb, and Mary Shaw. Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: a Case Study of OpenBSD. *Proc. METRICS,* Sep 2005, (to appear).

[18] Paul Luo Li, Jim Herbsleb, and Mary Shaw. ForecastingField Defects Using a Combined Time-based and Metrics-based Approach: a Case Study of OpenBSD. *CMUTechReport, CMU-ISRI-05-125,* 2005.

[19] Zhaohui Liu, Nalini Ravishanker, and Bonnie Ray. Modeling Dynamic Reliability Growth Using Bayesian Methods. *Reliability Review*, vol. 23 no. 1, Mar 2003, pp. 5-9.

[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. *Proc. ICSE,* May 2003, pp. 274-284.

[22] Audris Mockus, Ping Zhang, and Paul Luo Li. Predictors of Customer Perceived Quality. *Proc. ICSE,* May 2005, pp. 225-233.

[23] John Munson and Taghi Khoshgoftaar. The Dimensionality of Program Complexity. *Proc. ICSE,* May 1989, pp. 245-253.

[24] John Musa and Anthony Iannino and Kazuhira Okumoto. *Software Reliability*. McGraw-Hill, 1990.

[25] Magnus Ohlsson and Per Runeson. Experience from Replicating Empirical Studies on Prediction Models. *Proc. METRICS,* Jun 2002, pp. 217-226.

[26] Thomas Ostrand, Elaine Weyuker, and Thomas Bell. Where the Bugs are. *Proc. ISSTA,* vol. 29 no. 4, Jul 2004, pp. 86-96.

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

[28] Norman F. Schneidewind. Body of Knowledge for Software Quality Measurement. *IEEE Computer*, vol. 35 no. 2, Feb 2002, pp. 77-83.

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

[30] OpenBSD www.openbsd.org