11th International Software Metrics Symposium (METRICS 2005), Como, Italy, September, 2005

Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: 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 important components of many business software applications. Field defect predictions for open source software systems may allow organizations to make informed decisions regarding open source software components. In this paper, we remotely measure and analyze predictors (metrics available before release) mined from established data sources (the code repository and the request tracking system) as well as a novel source of data (mailing list archives) for nine releases of OpenBSD. First, we attempt to predict field defects by extending a software reliability model fitted to development defects. We find this approach to be infeasible, which motivates examining metrics-based field defect prediction. Then, we evaluate 139 predictors using established statistical methods: Kendall's rank correlation, Pearson's rank correlation, and forward AIC model selection. The metrics we collect include product metrics, development metrics, deployment and usage metrics, and software and hardware configurations metrics. We find the number of messages to the technical discussion mailing list during the development period (a deployment and usage metric captured from mailing list archives) to be the best predictor of field defects. Our work identifies predictors of field defects in commonly available data sources for open source software systems and is a step towards metrics-based field defect prediction for quantitatively-based decision making regarding open source software components.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – Process metrics, Product metrics, Software science
D.2.9 [**Software Engineering**]: Management – Software quality assurance

General Terms

Measurement, Documentation, Reliability, Experimentation

Keywords

Field defect prediction, open source software, reliability modeling, CVS repository, request tracking system, mailing list archives, deployment and usage metrics, software and hardware configurations metrics

1. INTRODUCTION

Open source software systems such as operating systems are important components of many business software applications. Being able to predict field defects (customer reported software problems requiring developer intervention to resolve) may allow existing quantitatively-based decision making methods to be used to:

1. Help organizations that are seeking to adopt open source software components to make informed choices between candidates
2. Help organizations using open source software components to decide whether they should adopt the latest release
3. Help organizations that adopt a release to better manage resources to deal with possible defects

In this paper, we present a case study of the open source operating system OpenBSD in which we try two different approaches to predicting field defects: model fitting and a metrics-based approach.

Prior work by Li et. al. [16] shows that the Weibull model is the preferred model for modeling the defect occurrence pattern of OpenBSD. In the work we report here, we attempt to predict field defects by extending a Weibull model from development to the field. We find that it is not possible to fit an acceptable Weibull model to development defects. The release dates of OpenBSD are consistently around the time when the rate of defect occurrences peaks. Hence, there is insufficient data to fit a Weibull model. This result is consistent with Kenny's finding in [7]. Our finding that it is not possible to fit a Weibull model until the rate of defect occurrences establishes the need for metrics-based field defect prediction.

Identifying and collecting predictors (metrics available before release) are pre-requisites activities for metrics-based field defect prediction. We attempt first steps toward a metrics-based field defect prediction model by identifying and collecting potentially important predictors of field defects for OpenBSD. Prior work has identified important predictors of field defects and has predicted field defects for commercial software systems (e.g. Khoshgoftaar et. al. [9], Ostrand et. al. [29], Mockus et. al. [22]). The categories of predictors used in prior work are product metrics, development metrics, deployment and usage (DU) metrics, and software and hardware configurations (SH) metrics. However, prior work has not examined open source software systems, has not examined all categories of predictors simultaneously, and has not identified commonly available data sources for each category of predictor. In this paper, we examine predictors of field defects for an open source software system.

Our experiments show it is possible to collect product, development, DU, and SH predictors from data sources commonly available for open source projects. We identify seven important predictors collected from mailing list archives and the CVS code repository. Somewhat surprisingly, the most important predictor for the OpenBSD project is the number of messages to the technical discussion mailing list during the development period, which is a deployment and usage metric collected from mailing list archives.

Section 2 discusses prior work and motivates our work. Section 3 describes OpenBSD. Section 4 and 5 discuss our data collection method and data analysis method. Section 6 presents the results. Section 7 contains a discussion of our findings. Section 8 is the conclusion.

2. PRIOR WORK AND MOTIVATION

In this section, we motivate our work by discussing prior work. We define *field defects* as user-reported code-related problems requiring programmer intervention to correct. This is the same definition used by Li et. al. in [16]. We discuss software reliability modeling, software metrics as predictors, and methods used to establish predictors as important.

2.1 Software reliability modeling

Prior work by Li et. al. [16] shows that it is possible to model the rate of field defect occurrences of OpenBSD using the Weibull model. Software reliability modeling research summarized by Lyu in [17] suggests that it may be possible to predict field defects by fitting a Weibull model to development defects and then extending the model to the field. This leads to our first question:

Is it possible to predict field defects by fitting a Weibull model to development defects and then extending the model to the field?

Li et. al. use non-linear least squares (NLS) regression to fit defect occurrence data to the Weibull model. We will use the same regression technique to fit a Weibull model to development defects.

2.2 Software metrics as predictors

Metrics are defined by Fenton and Pfleeger in [4] as outputs of measurements, where measurement is defined as the process by which values are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. Metrics available before release are *predictors*, which can be used to predict field defects.

We categorize predictors used in prior work using an augmented version of the categorization schemes used by Fenton and Pfleeger in [4], Khoshgoftaar et. al. in [15], and the IEEE standard for software quality metrics methodology [3]:

- Product metrics: metrics that measure the attributes of any intermediate or final product of the software development process [3]. The product metrics used in prior work are computed using a snapshot of the code. Product metrics have been shown to be important predictors by studies such as Khoshgoftaar et. al. [10], Jones et. al. [6], Khoshgoftaar et. al. [11], Khoshgoftaar et. al. [15], and Khoshgoftaar et. al. [13].
- Development metrics: metrics that measure attributes of the development process. The development metrics used in prior work are usually computed using information in the change management system or the CVS code repository. Development metrics have been shown to be important predictors by studies such as Khoshgoftaar et. al. [10], Khoshgoftaar et. al. [11], Khoshgoftaar et. al. [15], and Khoshgoftaar et. al. [13].
- Deployment and usage metrics (DU): metrics that measure attributes of deployment of the software system and usage in the field. Little prior work has examined DU metrics, and no data source is consistently used. DU metrics have been shown to be important predictors by studies such as Jones et. al. [6], Khoshgoftaar et. al. [11], Khoshgoftaar et. al. [15], Khoshgoftaar et. al. [13], and Mockus et. al. [22].
- 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. Little prior work has examined SH metrics and no data source is consistently used. SH metrics have been shown to be important predictors by Mockus et. al. [22].

Prior work has examined only commercial software systems, and no prior work has examined predictors in all the categories simultaneously. Product and development predictors commonly used in prior work can be computed for open source software systems since the data sources used to compute the predictors (e.g. snapshots of the code and information in the CVS code repositories) are commonly available for open source software projects. However, prior work examining DU and SH metrics has had accurate information about system deployment and the users, such as deployment logs and usage profiles (e.g. Khoshgoftaar et. al. [11]) or information from an customer monitoring system (Mockus et. al. [22]). The data sources used in those studies are not available for OpenBSD; therefore, the DU and SH metrics used in prior work are not available. These concerns lead us to two additional questions:

Is it possible to collect DU, and SH predictors using data sources commonly available for open source software projects?

What are the important predictors of field defects for OpenBSD?

2.3 Methods of establishing the importance of predictors

Three methods are commonly used to establish a predictor as important:

1. Show high correlation between the predictor and field defects. This method is recommended by IEEE [3] and is used by Ohlsson and Alberg [25] and Ostrand and Weyuker [28].
2. Show that the predictor is selected using a model selection method. This method is used by Jones et. al. [6] and Mockus et. al.[22].
3. Show that the accuracy of predictions improves with the predictor included in the prediction model. This method is used by Khoshgoftaar et. al. [10] and Jones et. al. [6].

We use methods 1 and 2 to determine important predictors in this paper. Since we examine predictors that may be included in a metrics-based field defect prediction model but do not actually produce a prediction model, we do not use method 3.

3. SYSTEM DESCRIPTION

In this section, we present the open source software system OpenBSD. We present project details, information on the code repository, information on the request tracking system, and information on the mailing list archives.

3.1 Project details

OpenBSD is a Unix-style operating system written primarily in C. The project dates back to 1995 and has developers (i.e. users who have the write access rights to the CVS code repository) in North America, South America, Europe, Australia, and Asia. This project is similar to the FreeBSD project examined by Dinh-Trong and Bieman [33].

We examine the project between approximately 1998 and 2004. During that time, there were 10 releases (of which we examine 9, as we explain below) and the CVS code repository documented development changes by 159 different developers.

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 [26]; 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 [26].

3.2 The code repository

The OpenBSD project manages its source code using a CVS code repository. Developers are users who have both read and write access rights. Someone becomes a "developer" (i.e. getting an account on the main server) by "doing some good work and showing that he/she can work with the team" [26]. Everyone else has read access to the CVS code repository.

3.3 The request tracking system

The OpenBSD project uses a problem tracking system. Anyone can report a problem by using the *sendbug* command built into OpenBSD [26]. Each problem report is assigned a unique number and stored in the bugs database. The problem report can be tracked on-line using the unique number. A problem report can be assigned one of four classes: sw-bug (software bug), doc-bug (documentation bug), change request, and support. All problem reports are initially marked as open, and then a developer acts on the report and changes the status accordingly.

Our measure of defects for OpenBSD is user submitted problem reports in the request tracking system of the class software bugs. We count each problem report (which may not be unique) because a user deemed the problem important enough to report. These software related problem reports require a developer's intervention to resolve. This measure of defects is used by Li et. al. in [16]. Defects that occur after the release date are considered field defects. Defects that occur during the development and test period are considered development defects.

3.4 Mailing lists

The OpenBSD project has 23 mailing lists in five categories:

- General interest lists
- Developer's lists

- Platform specific lists
- CVS changes mailing lists
- CTM (emails out deltas to the source).

Not all lists are active and not all lists are archived consistently. The two most complete archives are at sigmasoft [31] and MARC [18].

4. DATA COLLECTION

This section describes the data collection process we used to extract data from the request tracking system, the CVS repository, and the mailing lists. It also describes the predictors we collect.

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 date to the nearest month due to the time it takes to install the operating system, use the system, and discover and report a problem. Someone reporting a bug right after the un-rounded release date is unlikely to be using the latest release. This is the same approach taken by Mockus et. al. in [22].

We consider the date of the first reported defect rounded to the nearest month to be the start of development. The development period is then the duration between the start of development and the release date.

4.1 The request tracking system

We wrote Java programs and perl programs to download each problem report from the OpenBSD website and parse the report to extract the report open date, the class (e.g. sw-bug, doc-bug, or change request), the release reported against, and the machine (i.e. the hardware configuration such as i386 or sparc).

There was one anomaly. Three months of 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 is activity on the bugs mailing list, which indicates that problems did occur. This happened during development and deployment of release 3.2. As a result, we did not examine release 3.2.

4.2 The CVS repository

We used the CVS checkout command to download the tagged release version of the source code from the CVS repository for releases 2.4 to 3.3 (except release 3.2).

We used four metrics tools and several scripts to compute product metrics from the C source files. The tools we used were:

- RSM by M Squared Technologies [23]
- SourceMonitor by Campwood Software [1]
- c_count written by Thomas E. Dickey [2]
- metrics written by Brian Renaud [19]

We arrived at these tools by conducting a web search, asking experts for help, and posting to the comp.software-eng and comp.software.measurement newsgroups. We evaluated the collection of tools and selected those listed above.

We encountered an existing CVS bug when downloading the source code for release 2.4 and release 2.5. As a result, we had to bypass a directory that contained HTML help documents. We also encountered 10 files with coding anomalies that the metrics tools could not resolve. We skipped those files for all releases. These files constitute less than .1% of the number of C source files.

We used the CVS log command to obtain information on changes to the source code. We used the log information between the start of development of release 2.4 and the release date of release 3.3. There were 97,566 committed changes in the development periods of the nine releases.

4.3 Mailing list archives

We wrote java programs to extract the number of messages posted each month in the mailing lists archives.

Not all lists were archived consistently and not all lists were active. Consistent data was not available for many of the lists before 1998; therefore, we only considered releases after 1998. When an archive showed no messages for a certain month, we were often unable to determine if no messages were posted or if the archive failed to properly record messages (both of which occur). Therefore, lists that had intervals in which no messages were posted for more than three months were not considered.

4.4 Metrics

We provide a summary of the 139 predictors we collected. Due to space limitation, we do not present all the metrics.

4.4.1 Product metrics

We collected 101 product metrics using snapshots of the code from the CVS code repository. Due to tooling constraints, we did not collect all the product metrics used in the literature. However, we did collect metrics that covered all of the dimensions of variation in the product metrics identified by prior work. Munson and Khoshgoftaar identified the dimensions of product metrics (i.e. components of variance captured by product predictors) used in the literature using principal component analysis in [24]. Principal component analysis captures the dimensions of variance in a group of predictors. Predictors that load on the same principal component capture the same dimension of variance and are highly correlated with each other [24]; therefore, it may be sufficient to use a predictor from each dimension. We give the dimensions and examples of the product metrics we used to capture the variation in the dimension in table 1. The product metrics we used had been shown to load on the principal component by Munson and Khoshgoftaar in [24]

Table 1. Product metrics

Dimension	***Product metrics used in this study***
Control: metrics related the flow of program control	*Cyclo*: Cyclomatic complexity *KWbreak*: Number of occurrences of the key word break (which is equivalent to possible program knot count as shown by Khoshgoftaar and Szabo [14])
Action: number of distinct operations and statements	*UOpand*: Unique operands *UOpator*: Unique operators
Size: size or item count of a program	*Statements*: Total number of statements per file summed across all files *LOC*: Lines of code per file summed across all files
Effort: Halstead's effort metrics	*PGeffort*: Halstead's effort metric per file summed across all files
Modularity: degree of modularity of a program	*DeepNest*: Number of statements at nesting level >9 per file summed across all files

4.4.2 Development metrics

We collected 22 development metrics. Due to differences in the style of development, we were not able to collect the same development metrics used in the literature. However, we tried to collect metrics that captured the same intent as the metrics used in the literature in our study. We collected metrics that cover all of the independent dimensions of variation in the development metrics identified by Khoshgoftaar et. al. in [13] and [15] Khoshgoftaar et. al. used principal component analysis to identify the dimensions of variation in their development metrics in [13] and [15]. Khoshgoftaar et. al. examined a commercial software system while we examined an open source software system; therefore, we made changes to the metrics to account for the differences between commercial and open source software systems. We offer an interpretation of the dimensions captured by each principal component (which is not offered by Khoshgoftaar et. al.), examples of the metrics belonging to each dimension in [13] and [15], and the metrics we used to capture the same sources of variance in table 2. We made one major modification. Since OpenBSD did not distinguish between designers and testers, we combined the dimensions identified by Khoshgoftaar et. al. that separated designers and testers. We believe our metrics captured the same source of variation as the referenced metrics since the only changes we made were to accommodate differences between commercial and open source styles of development. (Metrics collected using the CVS code repository are indicated by 'CVS', ones collected using the request tracking system are indicated by 'RTS', and ones collected using mailing list archives are indicated by 'MLA'.)

Table 2. Development metrics

Dimensions [13] ***and*** [15]	***Example of metrics in dimensions*** [13]	***Development metrics used in this study***
Dimension 1: the number of changes	Total number of changes to the code for any reason	*TotalUpdate* (CVS): Total number of updates during the development period
Dimension 2: experience of the people making changes	Number of updates to this module by designers who had 10 or less total updates in entire company career	*BotHalfC* (CVS): Number of different developers making changes to files that are c source files during the development period who are in the bottom 50% of all developers ranked based on the number of changes
Dimension 3: amount of change to the code	Net increase in lines of code	*Difference* (CVS): Lines added to c source files minus lines deleted from c source files during the development period
Dimension 4 and 7: problems found during the development of the prior release	Number of problems fixed that were found by designers or during beta testing in the prior release	*PreBugsPrev* (RTS): Total number of field defects reported during the development period of the previous release
Dimension 5 : field problems found by customers in prior releases	Number of problems fixed that were found by customer in the prior release	*PreBugsAll* (RTS): Total number of field defects reported during the development period in all releases
Dimension 6 and 8: problems found during the development of the current release	Number of problems found by designers or during beta testing in the current release	*PreBugsCurrent* (RTS): Number of field defects reported against the release under development during the development period

4.4.3 Deployment and usage metrics

We collected nine deployment and usage metrics. The metrics we collected fall into 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 captured characteristics of deployment and usage because they quantified the amount of interest in OpenBSD, which might be related to how many systems were deployed and how much the systems were used. Request tracking predictors counted the number of problem reports during development that were not defects. We believed our request tracking system predictors captured characteristics of deployment and usage because users had to install OpenBSD and use the system before they could report a problem. We present the two categories, examples of predictors in the categories, and short justifications for the predictors in table 3.

Table 3. Deployment and usage metrics

Category of predictors	***DU metrics used in this study***	***Justification***
Mailing list predictors	*MiscMailings* (MLA): number of messages to the miscellaneous mailing list, a general interest mailing list, during the development period *AdvocayMailings* (MLA): number of messages to the advocacy mailing list (which promotes the use of OpenBSD), a general interest mailing list, during the development period	These metrics quantify the amount of interest in OpenBSD, which may be related to how many systems are deployed and how much the systems are used.
Request tracking system predictors	*ChangeRequests* (RTS): Number of change requests during the development period *DocBugs* (RTS): Number of reported documentation problems during the development period	These metrics quantify the amount of deployment and usage because users must install OpenBSD and use the system before they can request changes or report documentation problems

4.4.4 Software and hardware configurations metrics

We collected seven software and hardware configurations metrics in all. The metrics we collected fall into 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 captured characteristics of software and hardware configurations because they reflect 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 identify the type of hardware used. We believed our request tracking system predictors captured characteristics of software and hardware configurations because users had to install OpenBSD on the specified HW before they could report a problem. We present the two categories, examples of predictors in the categories, and short justifications of the predictors in table 4.

Table 4. Software and hardware configurations metrics

Category of predictors	***SH metrics used in this study***	***Justification***
Mailing list predictors	*SparcMailing* (MLA) Number of messages to the sparc hardware specific mailing list, a platform specific mailing list, during the development period	This metrics may reflect the amount of interest/activity related to the specific hardware, which may be related to how many of the specific hardware machines have OpenBSD installed.
Request tracking system predictors	*CurrentBSDBugs i386HW* (RTS): Number of field defects reported against the current release during the development period that identify the machine as type i386	These metrics may quantify the number of machines with specific HW that have OpenBSD installed since users must install and use the system to report a problem

5. DATA ANALYSIS

First, we attempted to fit Weibull models to development defects. We used NLS regression to fit the Weibull models. NLS is a widely used model fitting method discussed in detail by Lyu in [17].

Next, we computed the correlations between the predictors and field defects in order to identify important predictors. We did not consider predictors that did not vary since they cannot predict field defects (e.g. we discarded the predictor measuring the number of instances of the key work ‘struct’ in the code, which was zero for all releases). We computed Spearman’s rank correlation (ρ), Kendall’s rank correlation (τ), and the statistical significance of the correlations. These are standard ways of computing rank correlation. Holland and Wolfe [5] recommended using rank correlation when the data are not be normally distributed. We determined that the data were not normally distributed by examining data plots. Refer to Weisberg [35], Venable and Ripley [34], and Hollander and Wolfe [5] for detailed explanations of rank correlation.

Finally, we performed a forward AIC model selection to identify important predictors. The predictors selected using the forward AIC model selection method complement each other since each predictor improves the fit substantially (i.e. enough to overcome the AIC penalty) even with the other predictors already in the model. The forward AIC model selection method can be used to select a subset of predictors as a first step in a regression analysis even if the data is not normally distributed. Refer to Weisberg [35] for a detailed explanation. The model selection process usually continues until the AIC score does not improve with additional predictors; however, since we had 9 observations and 139 predictors, we stopped at three iterations to prevent over fitting. Similar model selection methods were used by Ostrand et. al. in [29] and Khoshgoftaar et. al. in [12].

For all our analysis, we used the open source statistical program R [30].

6. RESULTS

We present the results of fitting the Weibull model, evaluating the predictors using correlation, evaluating the predictors using forward AIC model selection, and comparing important predictors. We find that the number of messages to the technical mailing list during development is the best predictor.

6.1 Prediction using a software reliability model

We are not able to fit a Weibull model to development defects. The NLS model fitting procedure does not converge for any of the releases. Our finding that the modeling fitting procedure does not converge is consistent with Kenny's findings in [7], which show that it is not possible to fit a Weibull model until most of the defects have occurred (i.e. past the hump in the number of defects). A typical release with the release date indicated is in shown figure 1. In 7 out of 9 releases, the release date is within two months of the time the rate of defect occurrences peak. In 8 out of 9 releases, the release date is either within one month or before the time the rate of defect occurrences peak. We cannot predict field defects by fitting a Weibull model to development defects.

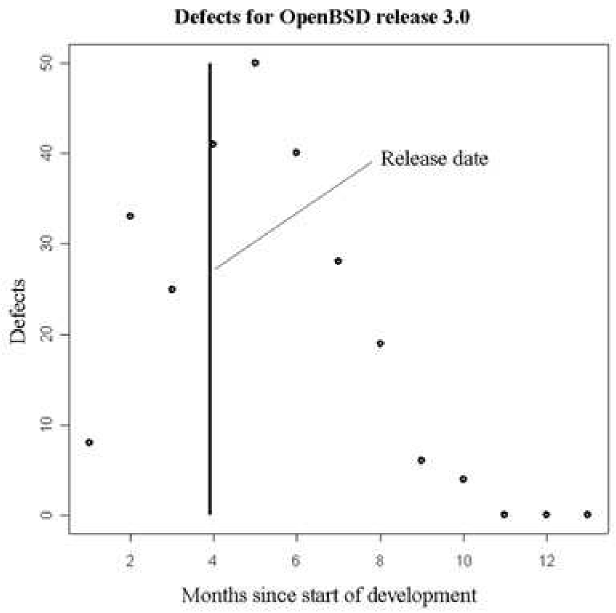

Figure 1. Defects for OpenBSD release 3.0

6.2 Analysis of predictors using correlations

Table 5 presents predictors of field defect occurrences that are significant at the 95% confidence level (CL) using rank correlation (a blank indicates that a predictor's correlation is not significant at the 95% CL). We briefly explain the predictors in this section.

Product metrics (computed using a snapshot of the code from the CVS code repository and the RSM metrics tool):

- *TotMeth:* Total number of methods.
- *PubMeth:* Number of public methods.
- *InlineComment*: Number of inline comments.
- *ProtMeth:* Number of protected methods.
- *CommentsClass:* Number of comments in classes summed across all classes.
- *InterfaceCompClass:* Number of parameters + number of returns in classes summed across all classes.
- *TotalParamClass:* Total number of parameters in classes summed across all classes.

Development metric (computed using the CVS code repository):

- *UpdateNotCFiles:* During the development period, the number of updates (deltas) to files that are not c source files.

Deployment and usage metric (computed using mailing list archives):

- *TechMailing:* Number of messages to the technical mailing list, a developer's mailing list, during development.

Software and hardware configuration metric (computed using mailing list archives):

- *SparcMailing:* number of messages to the sparc hardware specific mailing list, a platform specific mailing list, during the development period.

Table 5. Rank correlations

Predictor	*Kendall Correlation*	*p-value*	*Spearman Correlation*	*p-value*
TechMailing	0.61	0.02	0.78	0.02
TotMeth	0.61	0.02	0.73	0.03
PubMeth	0.61	0.02	0.73	0.03
CommentsClass	0.61	0.02	-	-
ProtMeth	0.57	0.03	0.67	0.05
InlineComment	0.56	0.04	0.68	0.05
InterfaceCompClass	0.51	0.05	-	-
TotalParamClass	0.51	0.05	-	-

6.3 Analysis of predictors using forward AIC model selection

We use three iterations of the forward AIC model selection method to select important predictors. Due to space limitation, we present the final linear model in table 6. The predictors are listed in the order selected. The AIC score of the final model is 75.52. The r^2 between the fitted model and field defects is 0.93. This high correlation suggests possible over fitting and confirms the need to stop at three iterations.

Table 6. AIC selected model

Variable	*Estimate coefficient*	*Std. Error*	*t value*	*Pr(>|t|)*
(Intercept)	134.32	18.06	7.437	0.0007
TechMailing	0.1102	0.015	7.445	0.0007
UpdatesNotCFiles	-0.0289	0.005	-5.757	0.0022
SparcMailing	0.1406	0.045	3.153	0.0253

The linear model in table 6 is not intended to be a valid prediction model. Additional steps need to be taken (e.g. adjust for non-constant variance) before the model can be used for prediction. Further validation of the predictors is also needed. We hope to do so in future work.

The estimated coefficients require interpretation. Since ranges of the predictors differ and all predictors are statistically significant, it is sensible to examine only the direction of the estimated coefficients (i.e. if they positive or negative). The coefficient for TechMailing is positive, indicating that increases in the metric correspond to more field defects. TechMailing measures the amount of deployment and usage of the system. This metric quantifies the amount of interest in OpenBSD, which may be related to how many systems are deployed and how much the systems are used. Our finding that increased deployment and usage correspond to more field defects is consistent with findings in Jones et. al. [6] and Mockus et. al. [22].

The coefficient for UpdatesNotCFiles is negative, indicating that increases correspond to fewer field defects. We think larger UpdatesNotCFiles may indicate maintenance (i.e. efforts to eliminate problems); therefore, it corresponds to fewer field defects. The coefficient for SparcMailing is positive indicating that increases in SparcMailing correspond to more field problems. Increase in SparcMailing may indicate increased activity/usage related to the sparc hardware, which may lead to field defects unaccounted for by the other predictors.

6.4 Comparison of important predictors

We compare the important predictors by examining the rank correlation among the predictors and field defects. This may allow us to determine which predictors may produce better predictions. We do not have enough observations to perform a principal component analysis.

The correlations between important predictors selected using rank correlation and field defects in table 7 indicate that increases in each of the predictors correspond to more field defects. These correlations are consistent with findings in prior work. The relationship between TechMailing (a DU metrics) and field defects is consistent with findings in Jones et. al. [6] and Mockus et. al. [22]. All other important predictors are product metrics. Our finding that increases in the product correspond to more field defects is consistent with findings in Ostand et. al.[29] and Jones et. al. [6]. However, the predictors are highly correlated with each other. This suggests that it may be sufficient to use just one of the predictors and that including all the predictors in a model may result in the multi-co-linearity problem discussed in Feton and Pfleeger [4] .

The correlation among important predictors selected using the forward AIC model selection method are lower than the correlation among important predictors selected using correlations. This confirms that the each predictor selected using model selection captures information not captured by the other predictors; therefore, they will complement each other in a prediction model and avoid the multi-co-linearity problem.

7. Discussion

We have established that it is not possible to fit a Weibull model to development defects for OpenBSD. We present results from fitting the Weibull model because prior work has identified the Weibull model as the preferred model. In addition, we also have results from experiments showing that it is not possible to make meaningful field defect predictions by extending other software reliability models fitted to development defects (i.e. the Gamma model, the Logarithmic model, the Exponential model, the Power model). Due to space limitations, those results are omitted. These results motivate the need to consider metrics-based field defect prediction.

We find that it is possible to collect product, development, DU, and SH predictors using data sources commonly available for open source software systems. In addition to validating the CVS code repository and the request tracking system as sources of predictors, we establish mailing list archives as an important data source, one not considered by previous studies.

Table 7. Correlations among important predictors

		AIC selected predictors			Correlation selected predictors			
	Field defects	*Sparc Mailing*	*Updates NotCFiles*	*Tech Mailing*	*Tot Meth*	*Pub Meth*	*Inline Comment*	*Prot Meth*
Field defects	1.000	0.278	-0.111	0.611	0.611	0.611	0.556	0.567
SparcMailing	0.278	1.000	0.500	0.111	0.556	0.556	0.278	0.433
UpdatesNotCFiles	-0.111	0.500	1.000	0.167	0.278	0.278	0.222	0.367
TechMailing	0.611	0.111	0.167	1.000	0.444	0.444	0.611	0.500
TotMeth	0.611	0.556	0.278	0.444	1.000	1.000	0.722	0.767
PubMeth	0.611	0.556	0.278	0.444	1.000	1.000	0.722	0.767
InlineComment	0.556	0.278	0.222	0.611	0.722	0.722	1.000	0.833
ProtMeth	0.567	0.433	0.367	0.500	0.767	0.767	0.833	1.000

We find that the most important predictor for the OpenBSD project is TechMailing collected from mailing list archives. The TechMailing predictor is the most highly rank correlated predictor with the number of defects and is the first variable selected using AIC forward model selection. We have validated this finding by talking with developers on the discussion forum. Feedback [27] indicates that this finding fits with the developers' intuition that participation by active developers (reflected by postings to the TechMailing list) leads to more defect discoveries. A plot of TechMailing against field defects is shown in figure 3. Other important predictors selected include four product metrics collected from the CVS code repository, a development metric collected from the CVS code repository (UpdatesNotCFiles), and a software and hardware configurations metric collected from mailing list archives (SparcMailings).

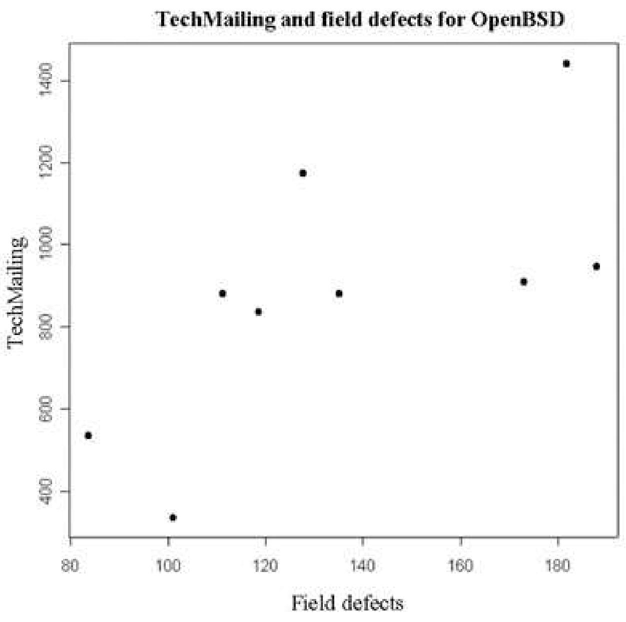

Figure 3. TechMailing and field defects

In contrast to findings in commercial software systems, (e.g. Mockus et. al. [21], Khoshgoftaar et. al. [10], and Khoshgoftaar et. al. [11]) predictors regarding changes to source files and those regarding developers are not important predictors for OpenBSD. We suspect this is due to the review and check-in process employed by the OpenBSD project (and possibly by other open source projects as well), which assures that all changes are of a certain quality regardless of the number of changes or the author of the change. All changes must be checked-in by a developer, who is someone that has shown ability to work on the code. This is supported by the explanation on the project webpage, which details how someone becomes a developer and gains the ability to check-in code. In addition, many changes are reviewed. We find evidence of this by observing logs of committed changes. Many logs contain markers (of the type "developer id" followed by the @ sign, e.g. art@) indicating that another developer has reviewed the changes.

8. CONCLUSION

In our case study of OpenBSD, we find that it is not possible to predict field defects by extending a Weibull model fitted to development defects. This indicates the importance of metrics-based field defect prediction models for open source software systems. We also find that it is possible to collect product, development, DU, and SH metrics using commonly available data sources for opens source projects. In addition, we identify important predictors that can be used to construct a field defect prediction model for OpenBSD using modeling methods in the literature. Such a model can help organizations make informed decisions regarding open source software components.

The paper presents novel and interesting findings, which are appropriate for a case study. However, our experiments need to be replicated on other open source projects. Replications can help verify that the relationships we have established are not due to chance alone. Future studies can include similar projects developing operating systems like FreeBSD or Debian and other types of systems like MySQL or JakartaTomcat.

Replication of our experiments is relatively straightforward since data sources we use are commonly available for open source software systems. For example, all projects hosted by SourceForge [32] use a CVS code repository, a request tracking system, and have mailing lists.

We do not consider non-c source files in our analysis (e.g. perl files and assembly files). These files may contain valuable information. However, since most of the system is written in c, we feel c source files are the most appropriate files to analyze. In release 3.4 (the most recent release we examine), there are ~36384 files in total. Approximately 17578 are c source files, 2378 are perl source files, and 1624 are assembly files. The remaining files are mostly documentation, configuration, and installation files.

There maybe other metrics we have failed to collect. For example, it may be possible to parse the defect reports for more detailed information regarding bugs, such as which software applications were running when the bug occurred. Since the data sources are available to everyone, we encourage others to explore other predictors.

Results in this paper represent a promising step towards quantitatively-based decision making regarding open source software components. The next step is to use the results in this paper and metrics-based modeling methods in the literature to construct metrics-based field defect prediction models and then to compare their predictions (e.g. the trees based method used by Khoshgotaar et. al. in [13], the neural networks method used by Khoshgoftaar et. al. in [14], and the linear regression used by Mockus et. al. in [22]).

9. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grants ITR-0086003, IIS-0414698, 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 thank the developers of OpenBSD for answering our postings. We thank the tool vendor who gave us trial licenses. We thank George Fairbanks, Patrick Riley, and Greg Wolfson for their help and insight.

10. REFERENCES

[1] Campwood Software. http://www.campwoodsw.com/

[2] c_count. http://dickey.his.com/c_count/c_count.html

[3] IEEE standard for a software quality metrics methodology. In *IEEE Std 1061-1998*, 1998.

[4] Norman Fenton and Shari Pfleeger. *Software Metrics - A Rigorous and Practical Approach*. Chapmann & Hall, London, 1997

[5] Myles Hollander and Douglas A. Wolfe. *Nonparametric statistical inference*. Wiley & Sons, 1973.

[6] Wendell Jones, John P. Hudepohl, Taghi M. Khoshgoftaar, and Edware B. Allen. Application of a Usage Profile in Software Quality Models. In *3rd European Conference on Software Maintenance and Reengineering*, 1999.

[7] Garrison Kenny. Estimating defects in commercial software during operational use. In *IEEE TR on Reliability*, 1993.

[8] Taghi M. Khoshgoftaar, Edward B. Allen, Wendell Jones, and John Hudepohl. Which software modules will have faults that will be discovered by customers? In *Journal of Software Maintenance: Research and Practice*, 1999

[9] Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, and Nitith Goel. Predictive modeling of software quality for very large telecommunications systems. In *IEEE International Conference on Communications*, 1996.

[10] Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, Nitith Goel, John Hedepohl, and Jean Mayrand. Detection of fault-prone program modules in a very large telecommunications system. In *Proceedings of ISSRE*, 1995.

[11] Taghi M. Khoshgoftaar, Edward B. Allen, Xiaojing Yuan, Wendell D. Jones, and John P. Hudepohl. Preparing measurements of legacy software for predicting operational faults. In *ICSM*, 1999.

[12] Taghi Khoshgoftaar, Adhijit Pandya, and David Lanning. Application of neural networks for predicting program faults. In *Annals of Software Engineering*, 1995.

[13] Taghi M. Khoshgoftaar, Ruqun Shan, and Edward B. Allen. Using product, process, and execution metrics to predict fault-prone software modules with classification trees. In *HASE*, 2000.

[14] Taghi Khoshgoftaar and Robert Szabo. Using neural networks to predict software faults during testing. In *IEEE Transaction on Reliability*, 1996.

[15] Taghi M. Khoshgoftaar and Vishal Thaker and Edward Allen. Modeling fault-prone modules of subsystems. In *Proceedings of ISSRE*, 2000.

[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. *FSE*, 2004.

[17] Michael Lyu. *Handbook of Software Reliability Engineering*. IEEE Society Press, USA, 1996.

[18] MARC. http://marc.theaimsgroup.com/

[19] metrics. http://www.chris-lott.org/resources/cmetrics/

[20] Audris Mockus, Roy Fielding, and James Herbsleb. A case study of open source software development: the Apache server. *ICSE*, 2000.

[21] Audris Mockus, David Weiss, and Ping Zhang. Understanding and predicting effort in software projects. In *ICSE*, 2003

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

[23] M Squared Technologies. http://msquaredtechnologies.com

[24] John Munson and Taghi Khoshgoftaar. The dimensionality of program complexity. In *ICSE*, 1989.

[25] Niclas Ohlsson and Hans Alberg. Predicting fault-prone software modules in telephone switches. In *IEEE Transactions on Software Engineering*, 1996

[26] OpenBSD. www.openbsd.org.

[27] OpenBSD discussion thread. http://marc.theaimsgroup.com/?t=110788031900001&r=1&w=2

[28] Thomas J. Ostrand and Elaine J. Weyuker. The Distribution of Faults in a Large Industrial Software System. In *ISSTA*, 2002.

[29] Thomas J. Ostrand and Elaine J. Weyuker and Thomas Robert M. Bell. Where the bugs are. In *ISSTA*, 2004.

[30] R. http://www.r-project.org/

[31] Sigmasoft. http://www.sigmasoft.com/~openbsd/

[32] SourceForge. http://sourceforge.net/

[33] Trimg Dinh-Trong and James M. Bieman, Open source software development: a case study of FreeBSD. *Metrics*, 2004.

[34] W.N. Venables and Brian D. Ripley. *Modern Applied Statistics with S-plus, 4th edition*. Springer-Verlag, 2000.

[35] Sanford Weisberg. *Applied Linear Regression, 2nd Edition*. Wiley and Son, 1985.

[36] Xiaohong Yuan, Taghi Khoshgoftaar, Edward Allen, and K Gasesan. An application of fuzzy clustering to software quality prediction. In *IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 2000.