An Investigation of the Relationships between Lines of Code and Defects

Hongyu Zhang[1, 2]

[1]School of Software, Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn

[2]State Key Laboratory for Novel Software Technology, Nanjing University
Nanjing 210093, China

Abstract

It is always desirable to understand the quality of a software system based on static code metrics. In this paper, we analyze the relationships between Lines of Code (LOC) and defects (including both pre-release and post-release defects). We confirm the ranking ability of LOC discovered by Fenton and Ohlsson. Furthermore, we find that the ranking ability of LOC can be formally described using Weibull functions. We can use defect density values calculated from a small percentage of largest modules to predict the number of total defects accurately. We also find that, given LOC we can predict the number of defective components reasonably well using typical classification techniques. We perform an extensive experiment using the public Eclipse dataset, and replicate the study using the NASA dataset. Our results confirm that simple static code attributes such as LOC can be useful predictors of software quality.

1. Introduction

It is commonly believed that there are relationships between external software characteristics (e.g., quality) and internal product attributes (e.g., size). Discovering such relationships has become one of the objectives of software metrics [3]. Many researchers believe in the merits of static code metrics and have proposed many quality prediction models based on them (e.g., [2, 6, 7, 8, 9, 21]), whereas some others are skeptical [5, 14].

Lines of Code (LOC) is a software size metric. It counts each physical source line of code in a program, excluding blank lines and comments. Although some authors have pointed out the deficiencies of LOC, it is still the most commonly used size measure in practices because of its simplicity.

In this paper, we investigate the relationship between LOC and defects, using the Eclipse project as a case study. We analyze the public Eclipse defect dataset, which includes defect data (pre-release defect data and post-release defect data) at both package-level and file-level, collected from three versions of the Eclipse system (v2.0, v2.1 and v3.0). Our findings are as follows:

- Through hypothesis testing, we find that there is sufficient statistical evidence that a weak but positive relationship exists between LOC and defects. Larger modules tend to have more defects.
- We confirmed the ranking ability of LOC [5]. When we order the modules (both packages and files) with respect to LOC, we find that a small number of largest modules accounts for a large proportion of the defects. For example, in Eclipse 3.0, 20% of the largest files are responsible for 62.29% pre-release defects and 60.62% post-release defects. Further analysis shows that the ranking ability of LOC [5] can be formally described using Weibull functions.
- We find that defect density (measured as defects/KLOC) obtained from a small percentage (e.g., 10%) of largest modules can predict the total number of defects well, at both package and file levels. For example, using the defect density value calculated from the top 10% largest Eclipse modules, we can estimate 70%-95% of the total number of defects.
- We are able to predict defective components (packages) based on LOC reasonably well. Using five typical machine-learning techniques, we construct classification models that classify a component to either defective (with one or more defects) or non-defective (with no defects). The 10-fold cross-validation results show that all five classification techniques can predict well. For example, for predicting pre-release defective components in Eclipse 3.0, the obtained recall is above 85% and precision is above 71%.

To achieve a better understanding of the general nature of our findings, we also perform a second case study on the NASA dataset. The replication study confirms our results obtained from the Eclipse dataset. Our experiments show that simple static code attributes such as LOC can be useful indicators of software quality.

978-1-4244-4828-9/09/$25.00 2009 IEEE Proc. ICSM 2009, Edmonton, Canada

2. Background

Defects are faults ("bugs") in programs that, when executed under particular conditions, causes failures in software system [3]. Over the years, there has been much research on quantitative analysis of defects. For example, Fenton and Ohlsson [5] analyzed pre-release defects (faults found during function and system tests) and post-release defects (faults found during site tests and one year of operation) in two successive releases of a large telecommunication system. They evaluated a range of basic hypotheses relating to defects and size. They found the evidence of Pareto-like distribution of defects, with a small number of modules containing a large proportion of defects. They also found that LOC is good at ranking the most defective modules, when the modules are ordered by LOC. They called it the "ranking ability" of LOC. Andersson and Runeson [1] replicated the study of Fenton and Ohlsson. Their results confirmed the Pareto principle of defect distribution, but did not conclusively support the LOC's ability in defect predictions. In our prior study, we discovered that the distribution of defects over modules can be better modeled as the Weibull distribution when the modules are ordered by the number of defects [19].

It is widely believed that some internal properties of software, such as size, have relationships with software quality. Many defect prediction models have been proposed based on the measurement of such properties. For example, Compton and Withrow [2] proposed a LOC-based polynomial regression model to predict the number of defects in a software system. They also proposed the "Goldilocks Principle" which claims that there is an optimal module size for minimizing defect density. While Fenton and Ohlsson [5] found it is not the case that size explains in any significant way the number of faults, and that there is no strong evidence that sizes metrics can be used as good indicators of defects. Fenton also showed in [4] that the "Goldilocks Principle" lacks support, and claimed that "the existing models are incapable of predicting defects accurately using size and complexity metrics alone". Other authors have found some but inconsistent correlations between the number of defects and module sizes [1].

There are also many classification models proposed to predict defect-proneness of modules from static code attributes. For example, Khoshgoftaar and Seliya [6] performed an extensive study on NASA dataset using 25 classification techniques with 21 code metrics including LOC. However, the prediction results are not satisfactory. A recent work of Menzies [9] reported improved probability of detection using the Naïve Bayes classifier, but we pointed out that their model is unsatisfactory when precision is concerned [16]. Many studies have shown that, when the prediction is made at a larger granularity level (such as component-level), the prediction accuracy is significantly improved [7, 18, 20]. El-Emam et al. [10] have demonstrated that size is a confounding factor when using object-oriented design measures to predict defect-proneness.

3. Data Collection and Analysis

In this research, we use the public Eclipse data (version 2) collected by the University of Saarland[1]. Eclipse is a widely used integrated development platform for creating Java, C++ and web applications. The public Eclipse dataset contains measurement and defect data for Eclipse versions 2.0, 2.1 and 3.0 (Table 1). The data was collected at two granularity levels: file and package levels. The studied Eclipse systems are considered large software systems - they contain in average 1030K LOC, 8403 classes and 491 packages.

The defect data is mined from Eclipse's bug databases and version achieves. There are two kinds of defects: pre-release defects (defects reported in the last six months before release) and post-release defects (defects reported in the first six months after release). Note that the number of defects of a package is different from the sum of defects of all files within the package. This is because in data collection only distinct defects are counted, which means that if the same defect occurs many times in several files within a package, it is only counted once for the package. More details about the data collection process can be found at [20].

Initial study of the Eclipse data shows that most modules (including both packages and files) have low measurement values with a few modules have large values. For example, Figure 1 shows the distribution of package size in Eclipse 3.0 (the packages are ranked according to their size, from the largest LOC to the smallest LOC). We can see that the distribution is highly skewed: that only a few packages have very large size, while most packages have small size. In the similar way, when we rank the number of defects (including both pre-release and post-release defects) of each module, we find that the distribution of defects is also highly skewed (Figure 1). Our prior studies show that the distribution of LOC follows the lognormal distribution [17] and the distribution of defects (when modules are ranked according to the number of defects) follows the Weibull distribution [19]. In this study, we explore the relationship between LOC and defects, especially the ability of LOC in defect prediction.

[1] http://www.st.cs.uni-sb.de/softevo/

Table 1. The Eclipse dataset

Version	Total LOC	Files	Packages	Pre-release Defects (package)	Post-release Defects (package)	Pre-release Defects (file)	Post-release Defects (file)
Eclipse 2.0	797K	6729	377	4297	917	7635	1692
Eclipse 2.1	987K	7888	434	2982	662	4975	1182
Eclipse 3.0	1306K	10593	661	4645	1534	7422	2679

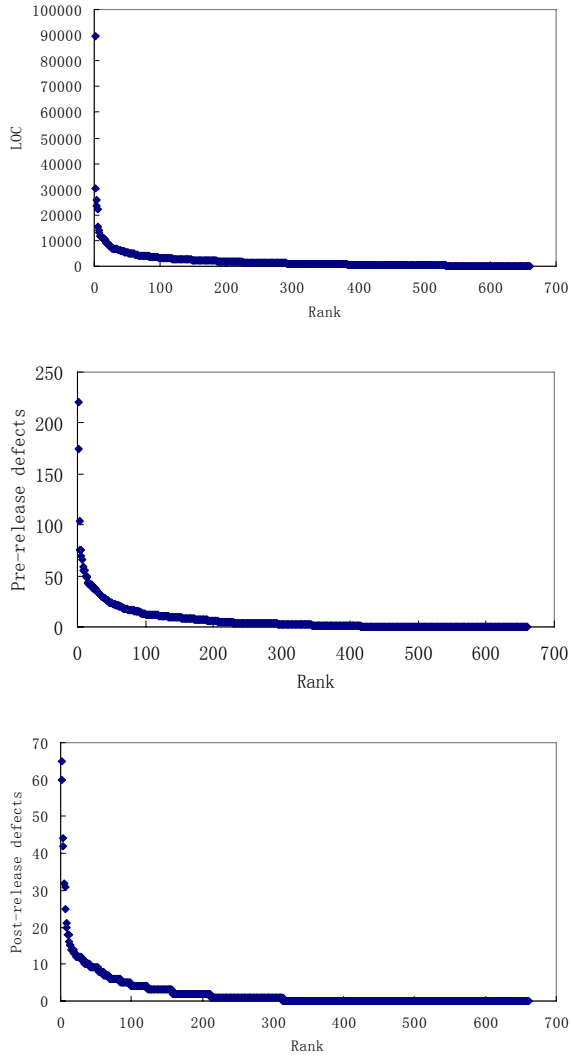

Figure 1. The distribution of LOC and Defects in Eclipse 3.0 (package level data, all packages are ranked according to their measurement values)

4. The Relationship between LOC and the Number of Defects

4.1 Correlation between LOC and Number of Defects

Having collected data, we statistically evaluate the correlation between LOC and the number of defects. The previous section shows that the LOC and defects data are not normally distributed, therefore we use the Spearman rank order correlation, a nonparametric method for measuring the relationship between variables.

For random variables X and Y, suppose we have pairs of observations (X_i, Y_i), *i=1..n*. Let $R(X_i)$ denote the rank of X_i among the X's and $R(Y_i)$ denote the rank of Y_i among the Y's. The Spearman rank order correlation, denoted r_s, is the correlation between $R(X_i)$ and $R(Y_i)$.

To statistically test if there is relationship between LOC and defects, we make the following hypotheses:

H0: there is no relationship between LOC and Number of Defects.
H1: there is relationship between LOC and Number of Defects.

If H0 is accepted, then there is no relationship between LOC and defects in the sense that all possible combinations of LOC and defects are equally like to occur. Using the statistical package SPSS, we obtain the results as show in Table 2. All r_s values are greater than 0. The test of Z statistic shows that all correlations are significant at the 0.01 level (2 tailed). Therefore, we reject the null hypothesis and conclude that there is a weak but positive relationship between LOC and defects. The result shows that larger modules tend to have more defects. The relationship is stronger in package level than in file level. This relationship motivated us to explore further the techniques for predicting defects based on LOC.

Table 2. The Spearman correlation r_s between LOC and the number of defects

	Pre-release		Post-release	
	Package	File	Package	File
Eclipse 3.0	0.581	0.421	0.559	0.333
Eclipse 2.1	0.575	0.429	0.471	0.259
Eclipse 2.0	0.585	0.385	0.564	0.357

* All correlations are significant at the 0.01 level (2-tailed)

4.2 The "Ranking Ability" of LOC

Many researchers have studied the relationship between LOC and defects. Fenton and Ohlsson [5] used diagrams (called Alberg diagrams) to evaluate the ability of LOC to identify the most defective modules. The Alberg diagrams plot the cumulative percentage of the number of defects when modules are ranked with respect to LOC. They found that LOC is quite good at ranking the most defective modules. They termed it the "ranking ability" of LOC.

In this paper, we also study the ranking ability of LOC for Eclipse systems. We rank the Eclipse modules according to their LOC (from the largest to the smallest), and compute the number of defects the top k (k = 5%, 10%, 15%, 20%) largest modules are responsible for. We find that a small number of largest modules accounts for a large proportion of the defects. For example, in Eclipse 3.0, 20% of the largest packages are responsible for 60.34% of the pre-release defects and 63.49% of post-release defects. At the file level, 20% of the largest Eclipse 3.0 files are responsible for 62.29% pre-release defects and 60.62% post-release defects. Figures 2 and 3 show the cumulative percentage of defects vs. the cumulative percentage of modules for both pre-release defects and post-release defects in Eclipse 3.0. Similar phenomenon is observed in other Eclipse versions (Table 3). Our results are consistent with those reported by Fenton and Ohlsson [5]. The results imply that, by simply ranking the modules according to their sizes, it is possible to locate a large number of defects.

Further analysis shows that the curves in Figures 2 and 3 (Alberg diagrams) follow the Weibull distribution. The CDF (cumulative density function) of the Weibull distribution [13] can be formally defined as:

$$P(x) = 1 - \exp\left(-\left(\frac{x}{\gamma}\right)^{\beta}\right) \qquad (\gamma > 0, \beta > 0).$$

Using statistical packages such as the SPSS, we are able to perform non-linear regression analysis and derive the parameters for each distribution. Figures 2 and 3 also show the fitted curves of Weibull distributions for Eclipse 3.0. Clearly the Weibull distribution fits the actual data well, at both package level and file level.

To statistically compare the goodness-of-fit of the Weibull distribution, we compute the coefficient of determination (R^2) and the Standard Error of Estimate (S_e). The R^2 statistic measures the percentage of variations that can be explained by the model. Its value is between 0 and 1, with higher value indicating a better fit. S_e is a measure of the absolute prediction error. The larger S_e indicates the larger prediction error. Table 4 shows the Weibull parameters and the accuracy measures for all Eclipse versions studied. The R^2 value ranges from 0.966 to 0.995, the S_e values range from 0.017 to 0.041. These results confirm that defects follow the Weibull distribution when modules are ordered by LOC.

Table 3. Percentage of defects contained by top *k*% largest modules in Eclipse

		Package Level				File Level			
		Top 5%	**Top 10%**	**Top 15%**	**Top 20%**	**Top 5%**	**Top 10%**	**Top 15%**	**Top 20%**
Pre-release defects	2.0	21.74%	33.65%	41.68%	51.64%	24.57%	37.01%	46.99%	53.48%
	2.1	27.60%	40.11%	47.25%	56.17%	28.82%	43.46%	53.97%	61.01%
	3.0	29.88%	43.40%	51.63%	60.34%	32.98%	46.28%	55.05%	62.29%
Post-release defects	2.0	29.55%	41.66%	48.53%	55.73%	34.16%	46.87%	55.73%	61.88%
	2.1	33.53%	43.81%	49.70%	56.04%	28.09%	40.52%	47.72%	54.31%
	3.0	34.16%	46.68%	57.56%	63.49%	29.97%	44.05%	52.41%	60.62%

Table 4. The Weibull distribution of Eclipse defects when modules are ordered by LOC

		Package Level				File Level			
		γ	**β**	$\mathbf{R^2}$	$\mathbf{S_e}$	γ	**β**	$\mathbf{R^2}$	$\mathbf{S_e}$
Pre-release defects	2.0	0.259	1.023	0.995	0.019	0.259	0.897	0.991	0.023
	2.1	0.236	0.889	0.987	0.027	0.207	0.830	0.995	0.017
	3.0	0.204	0.835	0.992	0.020	0.193	0.780	0.992	0.019
Post-release defects	2.0	0.233	0.834	0.981	0.031	0.190	0.811	0.986	0.026
	2.1	0.233	0.798	0.966	0.041	0.242	0.853	0.988	0.026
	3.0	0.182	0.779	0.989	0.023	0.203	0.827	0.993	0.019

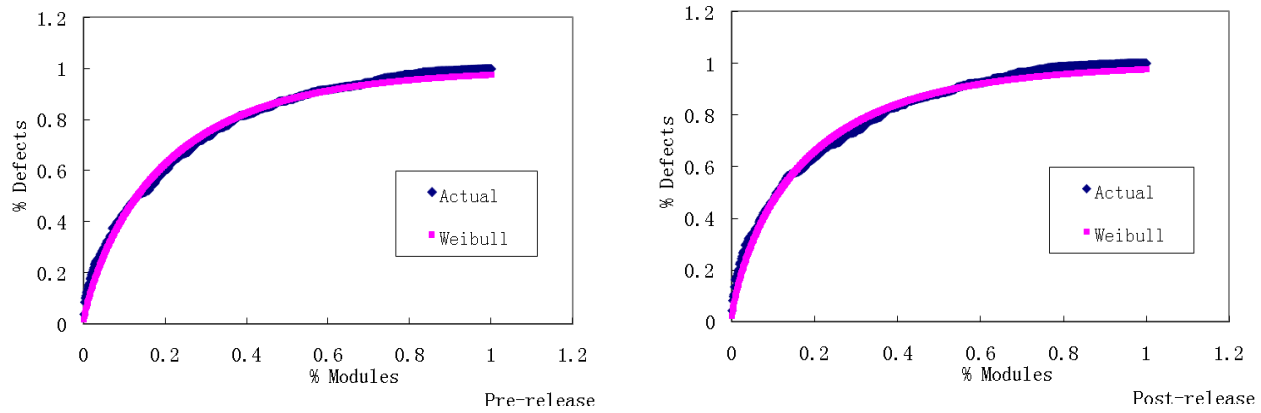

Figure 2: The Weibull distribution of defects when modules are ordered by LOC (Eclipse 3.0 package-level)

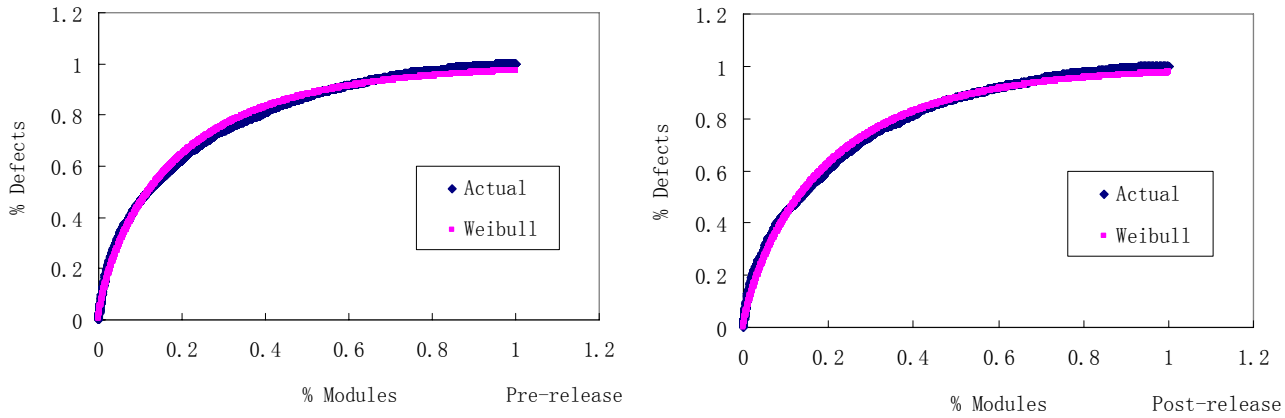

Figure 3: The Weibull distribution of defects when modules are ordered by LOC (Eclipse 3.0 file-level)

Table 5. Defect prediction based on *dd-10*% (defect density obtained from the top 10% largest modules)

		Package Level					File Level				
		Defect Density	#Actual defects	#Pred. defects	%	MRE	Defect Density	#Actual defects	#Pred. defects	%	MRE
Pre-release defects	2.0	3.78	4297	3011	70.07%	29.93%	7.22	7635	5756	75.39%	24.61%
	2.1	2.56	2982	2525	84.67%	15.33%	4.41	4975	4359	87.62%	12.38%
	3.0	3.17	4645	4140	89.13%	10.87%	5.20	7422	6790	91.48%	8.52%
Post-release defects	2.0	0.99	917	795	86.70%	13.30%	2.03	1692	1615	95.45%	4.55%
	2.1	0.62	662	612	92.45%	7.55%	0.98	1182	966	81.73%	18.27%
	3.0	1.13	1534	1470	95.83%	4.17%	1.78	2679	2332	87.05%	12.95%

Table 6. Defect prediction based on *dd-k%* for Eclipse 3.0

		Package Level				File Level			
		Defect Density	#Actual defects	#Predicted defects	MRE	Defect Density	#Actual defects	#Predicted defects	MRE
Pre-release defects	dd_5%	3.04	4645	3971	14.51%	5.24	7422	6843	7.80%
	dd_10%	3.17	4645	4140	10.87%	5.20	7422	6790	8.52%
	dd_15%	3.14	4645	4099	11.75%	5.15	7422	6728	9.35%
Post-release defects	dd_5%	1.15	1534	1499	2.28%	1.72	2679	2245	16.20%
	dd_10%	1.13	1534	1470	4.17%	1.78	2679	2332	12.95%
	dd_15%	1.16	1534	1510	1.56%	1.77	2679	2312	13.70%

4.3 Improving LOC's ability in defect prediction using defect density

To explore further LOC's ability in defect prediction, we examine the defect density of the top *k*% largest modules. Defect density has been recognized as a "de facto standard measure of software quality" [3]. We compute the defect density for the top *k*% largest modules (*dd_k*%) as: (the number of defects the top *k*% largest modules contain) / (the total KLOC of the top *k*% largest modules) * 100%.

After obtained the defect density values, we can then predict the total number of defects in the system. For example, for Eclipse 3.0 pre-release defects at package-level, the *dd_10*% (the defect density calculated from the top 10% largest Eclipse 3.0 packages) is 3.17 (about 3.17 defects per KLOC). The total size of Eclipse 3.0 is 1306 KLOC. We can then predict that the total number of pre-release defects is 4140. Comparing to the actual number of defects (4645), *dd_10*% achieves good prediction accuracy. This implies that by examining only 10% of the modules, we can estimate about 89.13% of the defects.

Table 5 shows the predictive power of *dd_10*% for all studied Eclipse versions. It gives the actual and predicted number of defects, and the percentages (%) of defects predicted.

We also evaluate the prediction accuracy using the MRE (Magnitude of Relative Error) measure, which is computed as (100% * |Predict-Actual|/Actual). A commonly accepted criteria is MRE<=25%. In Table 5, all MRE values (except for the case of pre-release defects in Eclipse 2.0) fall within the acceptable range (below 25%). For the case of pre-release defects in Eclipse 2.0, the MRE is just slightly above 25%. The results show we can achieve accurate predictions using the defect density measures calculated from the top 10% of largest modules.

We also experiment with different *k* levels (5%, 10% and 15%). In general all experimented defect density *dd_k%* can lead to satisfactory predictions for Eclipse systems (Table 6).

5. The Relationship between LOC and Defect-Proneness

In this section, we examine the ability of using LOC to predict the defect-proneness of components. Previous studies show that it is difficult to make predictions at a small granularity level (such as function/class level) due to the problem of imbalanced classification (the number of defective modules only accounts for a small percentage of the total modules) [16]. In this paper, we only examine the ability of LOC in predicting defective modules at the package (component) level. Each component is a high-level module that consists of a group of files. For the Eclipse systems, the number of defective components is shown in Table 7. We explore the effectiveness of identifying defective components based on LOC.

Table 7. The number of defective components (packages) in Eclipse dataset

	Eclipse 2.0	**Eclipse 2.1**	**Eclipse 3.0**
#of package	377	434	661
# of pre-release defective packages	283	287	415
# of post-release defective packages	190	194	313

5.1 Classifying defective components

Prediction of defective components can be cast as a classification problem in machine learning. A classification model can be learnt from the training samples of components with labels Defective (one or more defects) and Non-defective (no defects), the model is then used to classify unknown components.

We denote the defective components as the Positive (P) class and the Non-defective components as the Negative (N) class. A defect prediction model has four results: true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), as shown in Table 8.

Table 8. The results of a prediction model

Actual \ Predicted	Defective	Non-defective
Defective	TP	FN
Non-defective	FP	TN

In this project, we have chosen five commonly used classification techniques as shown in Table 9. All classifiers are supported by WEKA[2], a public domain data mining tool.

Before training the prediction models, we firstly use logarithmic filter to transform data *n* into their natural logarithms *ln(n)*. The transformation makes the data range narrower and makes it easy for classifiers to learn. We then construct the classification models using measurement data on LOC.

We use the 10-fold cross-validation to evaluate classification models. The whole training data is partitioned into 10 segments randomly. Each segment in turn is held as the test data and a classification model is trained on the rest nine segments.

Table 9. The classifiers used for defect prediction

Technique	Classifier in WEKA	Description
Multilayer Perceptron	Multilayer Perceptron	A backpropagation neural network
Logistic Regression	Logistic	A linear logistic regression based classification
Naive Bayes	NaiveBayes	A standard probabilistic Naive Bayes model
Decision Tree (C4.5)	J48	A C4.5 decision tree learner
K-Star	KStar	An instance based learning algorithm that uses entropy as the distance measure

5.2 Accuracy measures

To evaluate the prediction model, we use Recall and Precision, which are the accuracy measures widely used in Information Retrieval area. They are defined as follows:

$$\mathrm{Re}\,call = \frac{TP}{TP+FN}, \quad \mathrm{Pr}\,ecision = \frac{TP}{TP+FP}$$

[2] WEKA data mining tool is available at: http://www.cs.waikato.ac.nz/ml/weka/

The Recall defines the rate of true defective components in comparison to the total number of defective components, and the Precision relates the number of true defective components to the number of components predicted as defective. A good prediction model should achieve high Recall and high Precision. However, high Recall often comes at the cost of low Precision, and vice versa. Therefore, F-measure is often used to combine Recall and Precision. It is defined as the harmonic mean of Precision and Recall as follows:

$$F-measure = \frac{2 \times \mathrm{Re}\,call \times \mathrm{Pr}\,ecision}{\mathrm{Re}\,call + \mathrm{Pr}\,ecision}$$

The values of Recall, Precision and F-measure are between 0 and 1, the higher the better.

We also use the Accuracy metric (*Acc*, or *success rate* as termed in [15]) to complement F-measure to measure the overall accuracy of the prediction. The *Acc* is defined as follows:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN}$$

The *Acc* measure relates the number of correct classifications (true positives and true negatives) to the total number of instances.

5.3 Validation results

Table 10 shows the 10-fold cross validation results of the defect prediction models constructed using LOC (for Eclipse 3.0). For pre-release defective components, all classification techniques obtain good and consistent results with Recall above 85%, Precision about 71%, F-measure about 0.79, and *Acc* about 70%. These results are considered satisfactory.

For post-release data, the Recall values range from 67% to 77%, Precision ranges from 63% to 68%, the F-measure and *Acc* values are all close to 70%. We considered these results reasonably good. Similar results are obtained for other Eclipse versions. Although the prediction accuracy is still not perfect (especially for the post-release defects), the models can at least help managers make informed decision when allocating limited resources to test modules that seem more defect-prone.

6. The NASA Case Study

The experiments described in previous sections are performed over the Eclipse software systems. To achieve a better understanding of the general nature of the results, we conduct a replication study using the NASA dataset.

Table 10. Cross-Validation results of LOC-based classification models (Eclipse 3.0)

	Pre-release				Post-release			
Classifier	**Recall (%)**	**Precision (%)**	**F-measure**	**Acc (%)**	**Recall (%)**	**Precision (%)**	**F-measure**	**Acc (%)**
Multilayer Perceptron	85.5%	72.0%	0.78	70.0%	67.7%	66.5%	0.67	68.5%
Logistic Regression	86.7%	72.0%	0.79	70.5%	70.0%	67.6%	0.69	69.9%
Naive Bayes	89.4%	71.2%	0.79	70.7%	77.0%	65.0%	0.71	69.4%
Decision Tree	88.9%	71.7%	0.79	71.0%	71.2%	62.8%	0.67	66.4%
K-Star	87.5%	71.6%	0.79	70.3%	68.7%	67.0%	0.68	69.1%

The NASA dataset is stored in the NASA IV&V Facility Metrics Data Program (MDP) repository, which is available for public[3]. It contains software metrics data and defect data collected from many NASA projects (such as flight control, spacecraft instrument, storage management, and scientific data processing). In this study, we analyze 9 NASA projects, which are developed by C, C++ and Java languages, containing a total of 268K modules (function level), 265K LOC and 1716 defects (Table 11).

Descriptive data analysis shows that the distributions of NASA LOC and defect data are also highly skewed, with a few modules having large measurement values and a large number of modules having small values. The Spearman correlations show that there is a weak but positive relationship between LOC and defect numbers. We also confirm the ranking ability of LOC for NASA datasets -- that a small percentage of the largest modules contain a large percentage of defects (e.g., 20% of the largest modules contain 51%-100% defects). Further analysis show that, when the modules are ordered by LOC, the distribution of defects follows the Weibull function (Table 12). In Table 12, the R^2 values range from 0.948 to 0.998 and the S_e values range from 0.011 to 0.043. These results confirm our findings reported in Section 4.

Table 11. The NASA dataset

Project	**Total LOC**	**#Com-ponents**	**#Func-tions**	**# Defects**	**Lang.**
CM1	17K	20	16903	70	C
PC1	26K	29	25922	139	C
PC2	25K	10	26863	26	C
PC3	36K	29	36473	259	C
PC4	30K	33	30055	367	C
KC1	43K	18	42963	525	C++
MC1	66K	36	66583	79	C++
MC2	6K	1	6134	113	C++
KC3	8K	5	7749	101	Java
Total	**265K**	**182**	**267986**	**1716**	

[3] NASA dataset: http://mdp.ivv.nasa.gov/

Table 12. The Weibull distribution of NASA defects when the modules are ordered by LOC

Project	γ	**β**	$\mathbf{R^2}$	$\mathbf{S_e}$
CM1	0.251	1.095	0.988	0.029
PC1	0.193	0.754	0.981	0.029
PC2	0.073	0.603	0.948	0.036
PC3	0.234	1.089	0.998	0.011
PC4	0.194	0.957	0.995	0.017
KC1	0.236	1.015	0.986	0.030
MC1	0.080	0.819	0.979	0.026
MC2	0.305	0.996	0.973	0.043
KC3	0.174	0.895	0.976	0.035

Table 13. Predicting the total number of defects using *dd_10%*

Project	**#Actual defects**	**#Predicted defects**	**MRE**	**MRE <=25%?**
CM1	70	67	4.29%	yes
PC1	139	161	15.83%	yes
PC2	26	35	34.62%	no
PC3	259	219	15.44%	yes
PC4	367	355	3.27%	yes
KC1	525	440	16.19%	yes
MC1	79	102	29.11%	no
MC2	113	76	32.74%	no
KC3	101	102	0.99%	yes

As in the case of Eclipse, we can predict the total number of defects based on defect density obtained from the top *k*% of largest modules. In the replication study, we use *dd_10*% to predict the number of defects in each NASA project. The prediction results are shown in Table 13. All MRE values are within acceptable range (below 25%), except for the projects PC2, MC1 and MC2. For the project MC1 and MC2, the MRE values are just slightly above the acceptable criteria. For project PC2, the larger MRE value is due to the relative small actual value. The limitation of MRE measure for small numbers is already well known [11]. Actually, the absolute prediction error for PC2 is rather small (only 9). In Table 13, the average MRE

value is only 16.94%. Therefore, we conclude that the replication results support the validity of our defect prediction method described in Section 4.

The original NASA data is collected from modules at function/method level. To replicate the experiment for predicting defective components, we also identify the components in NASA dataset. We find that each NASA dataset includes a *Product Hierarchy* document that describes the function-component relationship in a project. A unique ID is assigned to each component and function. We develop a tool that can analyze the Product Hierarchy document and aggregate the function level data into component level data. Some NASA projects contain reused components, which cause duplicate records in the dataset; therefore we also preprocess the data to remove such duplication. For the 10 NASA projects we analyzed, total 182 components are identified. Among them, 81 components (44.51%) are defective.

Table 14 shows the 10-fold cross validation results of the defect prediction models constructed using LOC. The classification techniques obtain results with Recall ranging from 64.2% to 90.1%, Precision ranging from 60.3% to 65.8%, F-measure ranging from 0.65 to 0.72, and *Acc* ranging from 68.7% to 70.9%. These results are considered reasonably good, confirming the usefulness of our defect-proneness prediction method described in Section 5.

Table 14. Validation results for NASA dataset

Classifier	Recall (%)	Precision (%)	F-measure	Acc (%)
Multilayer Perceptron	66.7	64.3	0.66	68.7
Logistic Regression	64.2	65.8	0.65	69.2
Naive Bayes	79.0	62.7	0.70	69.8
Decision Tree	90.1	60.3	0.72	69.2
K-Star	77.8	64.3	0.70	70.9

7. Related Work and Discussions

Discussions on code metrics based defect prediction

Andersson and Runeson [1] studied three telecommunications projects. For these projects, they found 20% of the largest modules are responsible for 26%, 18% and 57% of the defects, respectively. The results for the first two projects showed that LOC was not good at ranking the most defective modules, while the third project indicated a better ranking ability of LOC. Ostrand et al. [12] also studied the ranking ability of LOC for an inventory system and a provisioning system. They found that 20% of largest files contained 73% and 74% of the defects for the two systems. In this paper, we confirm the ranking ability of LOC for the Eclipse and NASA datasets (e.g., 20% of the largest Eclipse modules are responsible for 51%-63% of the defects). We also discover that the ranking ability curve can be formally described using Weibull functions. Furthermore, using defect density values calculated from a small percentage of the largest modules, we can achieve a better prediction of the total number of defects (e.g., for Eclipse 3.0, the obtained MRE values are between 1.56% and 16.20%).

Many researchers have constructed models to predict defect-proneness based on the NASA dataset. For example, Menzies et al. [8] performed experiments on function/method level defect prediction on five NASA dataset. They obtained mean probability of detection (i.e. Recall) 36%. Khoshgoftaar and Seliya [6] performed an extensive study on the NASA JM1 and KC2 dataset using 25 classification techniques with 21 static code metrics. They also observed low prediction performance, and they did not see much improvement by using different classification techniques.

We believe that a major reason of having low prediction performance is the large number of small modules. The original NASA data is collected from modules at function/method level. The number of defective functions/methods only accounts for 0.44% of total functions/methods, which causes the difficulty of classification learning in the presence of imbalanced class distribution [16]. In addition, the size and other complexity measures for a single function/method are usually small, which makes it difficult for a machine learning technique to distinguish between defective modules and non-defective modules. This problem is also observed by Koru and Liu [7], who suggested stratifying dataset according to modules size.

In our research, we predict the defective modules at the component (package) level to avoid the above-mentioned problem. In our classification model, we only use LOC as the input. We tested our model on both Eclipse and NASA datasets, and the results are satisfactory (e.g., for the NASA dataset, the Recall ranges from 64.2% to 90.1%, and the Precision ranges from 60.3% to 65.8%).

Threats to validity

The results of this research are obtained from experiments on the Eclipse and NASA defect datasets. If these datasets are seriously flawed (e.g., there were major problems in bug data collection and recording), our results may be invalid. It is therefore desirable to replicate this study on more industrial datasets and to evaluate its validity. This is one of our important future work.

The other threat to validity is that, to ensure proper statistical analysis and to ensure the cost-effectiveness of the proposed method, the number of modules in the system should be large enough. Therefore, the proposed method is suitable for large-scale software systems.

In a well-managed project, all developers share similar development process and quality assurances procedures, therefore it is possible to achieve good and consistent defect prediction results. The applicability of defect prediction methods for bad-managed projects remains to be verified.

8. Conclusions

The use of static code attributes such as LOC for defect prediction has been widely debated. In this paper, we investigate the relationships between LOC and defects. We analyzed two public defect datasets: the Eclipse dataset and the NASA dataset. We confirm the "ranking ability" of LOC found by Fenton and Ohlsson, and discover that this ability can be actually modelled by a Weibull distribution function. We find that by using defect density values calculated from a small percentage of the largest modules, we can improve LOC's ability to predict the number of defects. In this investigation, we also find that using typical classification techniques, we are able to predict defective components reasonably well based on LOC.

LOC is one of the simplest software metrics and is cheap to collect. Our results show that LOC can be a useful indicator of software quality, and that we are able to build useful defect prediction models using LOC. In future, we will further evaluate our results for a variety of large-scale industrial projects. We hope our work could help achieve a better understanding of the relationship between static code metrics and software quality.

Acknowledgments

This research is supported by the Chinese NSF grants 90718022 and 60703060, and by the MOE Key Laboratory of High Confidence Software Technologies at Peking University.

References

[1] C. Andersson and P. Runeson, A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems, *IEEE Trans. Software Eng.*, 33 (5), pp. 273-286, May 2007

[2] T. Compton, and C. Withrow, Prediction and Control of Ada Software Defects, *J. Systems and Software*, vol. 12, pp. 199-207, 1990.

[3] N. Fenton and S. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, PWS, 1997.

[4] N. Fenton and M. Neil, A Critique of Software Defect Prediction Models, *IEEE Trans. Software Eng.*, 25 (3), pp. 675-689, 1999.

[5] N. Fenton and N. Ohlsson, Quantitative Analysis of Faults and Failures in a Complex Software System, *IEEE Trans. Software Eng.*, 26 (8), pp. 797-814, 2000.

[6] T.M. Khoshgoftaar and N. Seliya, The Necessity of Assuring Quality in Software Measurement Data, *Proc. 10th Int'l Symp. Software Metrics* (METRICS'04), IEEE Press, pp. 119–130, 2004.

[7] A. Koru and H. Liu, Building Effective Defect-Prediction Models in Practice, *IEEE Software*, vol. 22, no.6, pp. 23-29, 2005.

[8] T. Menzies, J. DiStefano, A. Orrego, and R. Chapman, Assessing Predictors of Software Defects, *Proc. Workshop Predictive Software Models*, 2004.

[9] T. Menzies, J. Greenwald and A. Frank, Data Mining Static Code Attributes to Learn Defect Predictors, *IEEE Trans. Software Eng.*, vol. 32, no. 11, 2007.

[10] K. El-Emam, S. Benlarbi, N. Goel, and S. Rai, The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics, *IEEE Trans. Software Eng.*, vol. 27, no. 6, pp. 630-650, July 2001.

[11] I. Myrtveit, E. Stensrud and M. Shepperd, Reliability and Validity in Comparative Studies of Software Prediction Models, *IEEE Trans. Software Eng.*, 31 (5), pp. 380-391, 2005.

[12] T. Ostrand, E. Weyuker and R. Bell, Predicting the Location and Number of Faults in Large Software Systems, *IEEE Trans. Software Eng.*, 31 (4), 2005.

[13] R. Ramakumar, *Engineering Reliability: fundamentals and applications*, Prentice-Hall, 1993.

[14] M. Shepperd and D. Ince, *Derivation and Validation of Software Metrics, Oxford University Press*, 1993.

[15] H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation,* second ed. Morgan Kaufmann, 2005.

[16] H. Zhang and X. Zhang, Comments on "Data Mining Static Code Attributes to Learn Defect Predictors", *IEEE Trans. on Software Eng.*, vol. 33(9), Sep 2007.

[17] H. Zhang and H. B. K. Tan, An Empirical Study of Class Sizes for Large Java Systems, *Proc. of 14th Asia-Pacific Software Engineering Conference (APSEC 2007)*, Nagoya, Japan, 2007. IEEE Press, pp. 230-237.

[18] H. Zhang, X. Zhang, M. Gu, Predicting Defective Software Components from Code Complexity Measures, *Proc. 13th IEEE Pacific Rim Int'l Symp. on Dependable Computing Conference (PRDC 2007)*, Melbourne, Australia, Dec 2007, IEEE Press, pp. 93-96.

[19] H. Zhang, On the Distribution of Software Faults, *IEEE Trans. on Software Eng.*, 34(2), 2008.

[20] T. Zimmermann, R. Premraj and A. Zeller, Predicting Defects for Eclipse, *Proc. 3rd Int'l Workshop on Predictor Models in Software Eng.*, 2007. The Eclipse data is available at http://www.st.cs.uni-sb.de/softevo/.

[21] T. Zimmermann and N. Nagappan, Predicting Defects using Network Analysis on Dependency Graphs, *Proc. 30th IEEE International Conference on Software Engineering (ICSE 2008),* Leipzig, Germany, 2008.