Software Defect Prediction based on LOC

Hongyu Zhang
School of Software, Tsinghua University
Beijing 100084, China
hongyu@tsinghua.edu.cn

Abstract

The ability to predict the number of defects in a software system can help us better understand and control software quality. In this paper, we analyze the relationships between Lines of Code (LOC) and defects (including both pre-release and post-release defects). We find that, when the modules are ranked according to their LOC, the number of defects across modules follows the Weibull distribution. Having understood the nature of defect distribution, we then propose a defect prediction method that can predict the total number of defects based on the Weibull model constructed from similar projects. 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 static code attributes such as LOC can be useful predictor 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]), whereas some others remain skeptical [5, 14].

Line 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 use the Eclipse project as a case study. We analyze the public Eclipse defect dataset, which include defect data (both pre-release defect data and post-release defect data) at both package-level and file-level from three versions of the Eclipse system (v2.0, v2.1 and v3.0). Through hypothesis testing, we find that there is sufficient statistical evidence that there is a weak but positive relationship between LOC and defects.

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 number of defects across ranked modules follows the Weibull distribution, a statistical distribution that is widely used in reliability engineering.

We also find that the Weibull distribution models for similar projects developed by the same organization are quite similar. Therefore, it is possible to construct a defect prediction model based on the Weibull model built from an organization's historic data. For example, using the Weibull model built from Eclipse 2.0, we can predict the total number of defects in Eclipse 3.0 after obtaining the defect data from top 20% of the largest modules. The relative prediction error is low (10.61% for pre-release defects and 9.79% for post-release defects).

In our experiments, we also find that 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 reasonably well.

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 static code attributes, such as LOC, can be useful indicators of software quality, and we can build effective prediction models based on LOC.

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 commercial 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 defect-prone modules, when the modules are ordered with respect to 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 ranking ability on defect-proneness. In our prior study, we discovered that the distribution of defects over modules can be better modeled as the Weibull distribution [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 [4] 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 Zhangs [16] pointed out that their model is unsatisfactory when precision is concerned. 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

3.1 The Eclipse dataset

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 contain 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 1030KLOC, 8403 classes and 491 packages.

The defect data is collected 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 time 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, and utilize the discovered relationship for 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

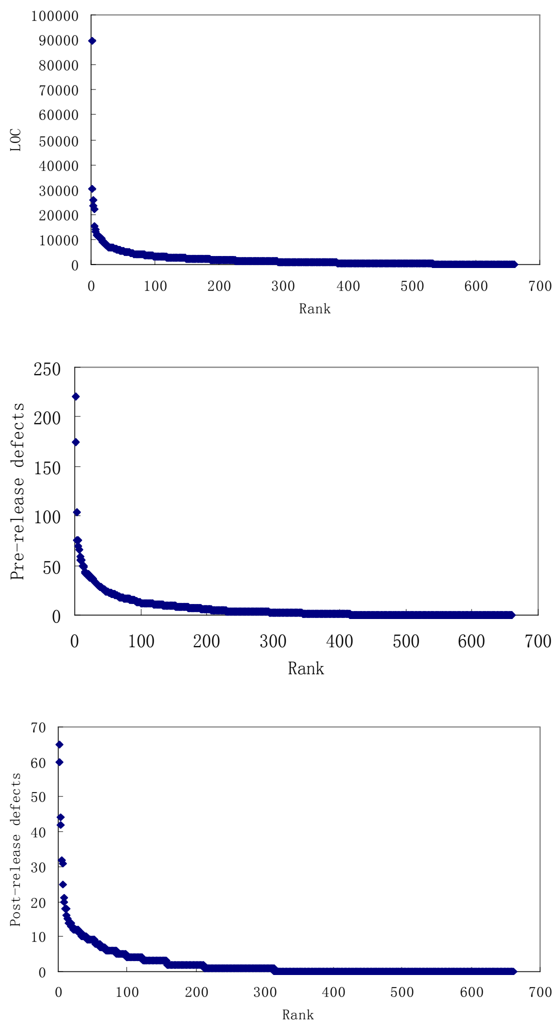

Figure 1. The distribution of LOC and Defects in Eclipse 3.0 (package level data, all packages are ranked according to their measurement values)

3.2 Correlation between LOC and Number of Defects

Having collected data, we then statistically evaluate the relationship 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 the ranks of the X's and Y's, and is defined as follows:

$$r_s = 1 - \frac{6 \sum_{i=1}^{n} \left[R(X_i) - R(Y_i) \right]^2}{n(n^2 - 1)}$$

The value of r_s ranges from 0 to 1. If there is no association between the X's and Y's, the expected value of r_s is 0. If the ranks of the Y's aligns perfecting with the ranks of X's, then r_s is 1. After r_s is computed, the Z statistic can be used to determine the statistical significance:

$$Z = \frac{r_s}{\sqrt{\mathrm{var}(r_s)}} = r_s \sqrt{n-1}$$

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 $\boldsymbol{r_s}$ 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 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. The Weibull Distribution of Software Defects when Modules are Ordered by LOC

For each Eclipse project, we rank the modules according to their size (from the largest LOC to the smallest LOC), and then calculate the cumulative percentage of defects. We find that the distribution is highly skewed - that a small number of largest modules accounts for a large proportion of the defects. For example, Figure 2 and 3 show the relationship between the cumulative percentage of modules and the cumulative percentage of defects in Eclipse 3.0, at package level and file level respectively. 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. Similar phenomenon is observed in other Eclipse versions. Our results are consistent with those reported by Fenton and Ohlsson [5]. The results show that, by simply ranking the modules according to their sizes, a large number of defects can be located. The results also suggest that, in software quality assurance practices (such as inspection, formal technical meeting, and testing), initial efforts can be centered on the largest modules so that a large number of defects could be identified.

Further analysis shows that the defects follow the Weibull distribution, when the modules are ordered by LOC. Weibull distribution, developed by the physicist Waloddi Weibull, is one of the most widely used probability distributions in the reliability engineering discipline [13]. The CDF (cumulative density function) of the Weibull distribution 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. Figure 2 and 3 show the fitted curves of Weibull distribution for both pre-release defects and post-release defects in 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 and is computed as:

$$S_e = \sqrt{\frac{\sum (y - y')^2}{n-2}},$$

where y and y' are the actual and predicted values, respectively. The larger S_e indicates the larger prediction error. For example, in Figure 2 the Weibull distribution has the R^2 value 0.992 for pre-release defects, meaning that the model accounts for 99.2% of the variations. The corresponding S_e value is 0.02, which represents relatively small variation. Table 3 below shows the Weibull parameters and the accuracy measures for all Eclipse versions studied. The R^2 value ranges from 0.966 to 0.992, 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. The Weibull distribution of Eclipse defects when modules are ordered by LOC

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

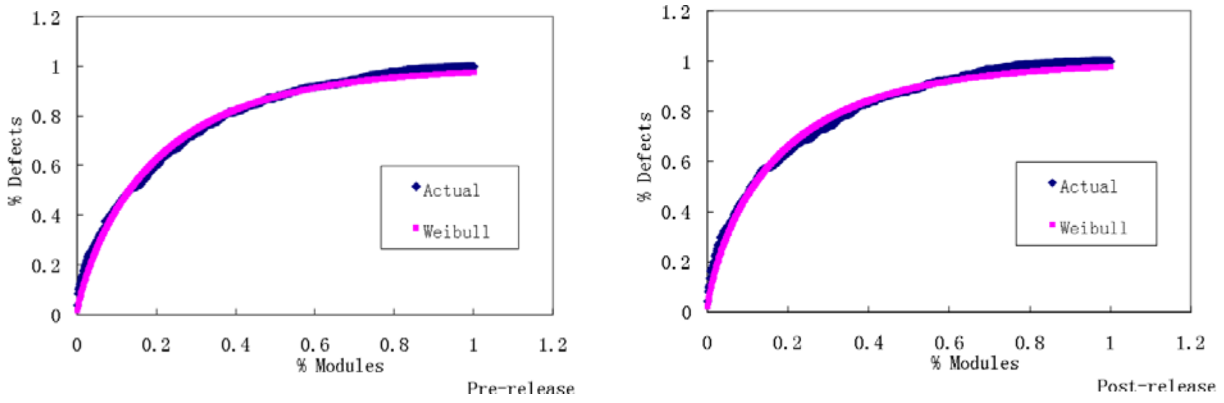

Figure 2: The Weibull distribution of defects when modules are ordered by LOC (Eclipse 3.0 package-level)

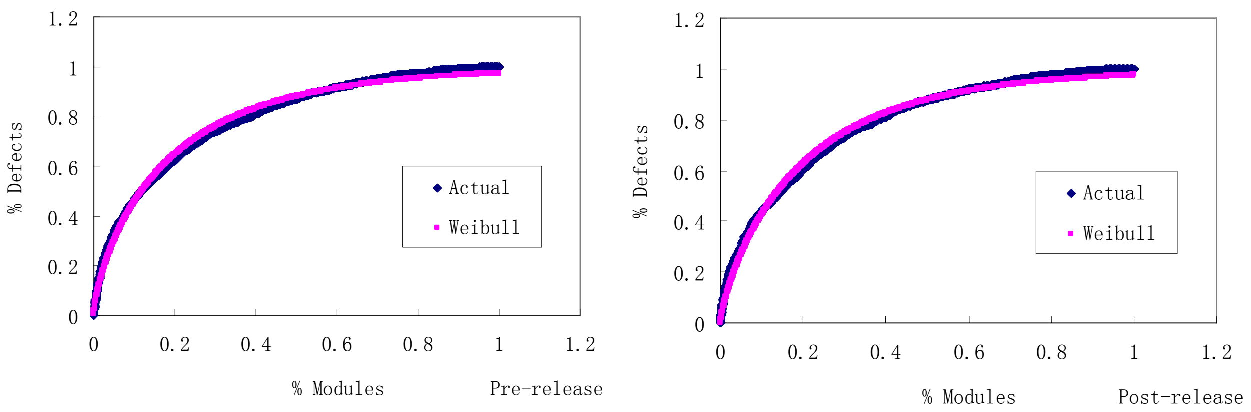

Figure 3: The Weibull distribution of defects when modules are ordered by LOC (Eclipse 3.0 file-level)

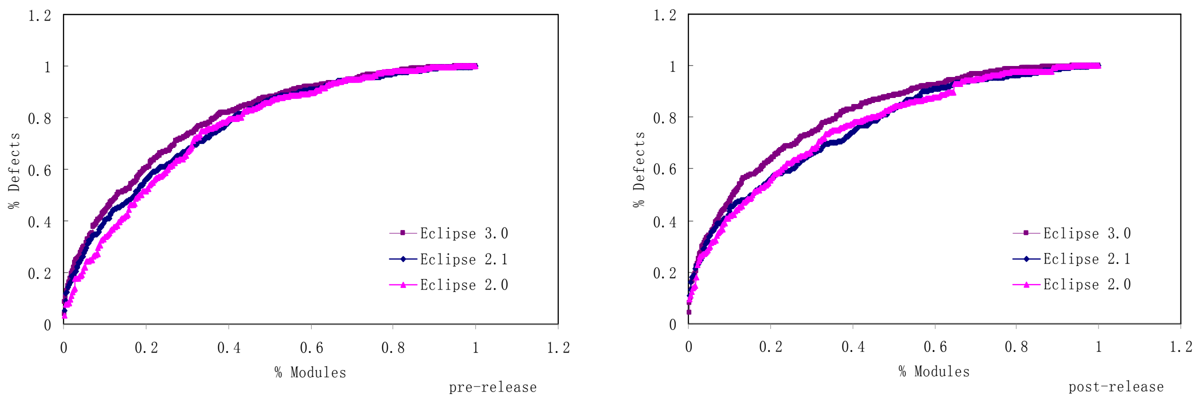

Figure 4. Defect distributions in Eclipse versions (package-level)

Table 4. Predicting the total number of defects based on data from the top 20% largest modules and the Weibull model built from Eclipse 2.0

		Package Level				File Level			
		#Actual defects	#Predicted defects	MRE	MRE <=25%?	#Actual defects	#Predicted defects	MRE	MRE <=25%?
Pre-release faults	Eclipse 2.1	2982	3063	2.72%	yes	4975	5350	7.54%	yes
	Eclipse 3.0	4645	5138	10.61%	yes	7422	8149	9.79%	yes
Post-release faults	Eclipse 2.1	662	611	7.70%	yes	1182	970	17.94%	yes
	Eclipse 3.0	1534	1608	4.82%	yes	2679	2455	8.36%	yes

5. Predicting the Number of Defects

Further analysis show that the Weibull based defect distribution models constructed for different versions of a system are quite similar. Figure 4 shows the distributions of defects in three Eclipse versions (modules are ordered by LOC). The distribution curves share similar trend and the maximum difference between any two curves is less than 15%. As the projects developed by the same organization often share similar coding standard, development process, testing procedure, and quality assurances measures, therefore it is not surprise to see that the distributions for different versions are in fact quite similar.

The similarity between defect distribution curves motivated us to predict defects based on an organization's historic data. Given a Weibull model built for a previous version, we can predict the total number of defects for a new version once we have initial test results for some largest modules. As an example, say after testing the 20% largest packages in Eclipse 3.0, we identify *m* defects. We can then predict the total number of defects *n* in Eclipse 3.0 using the Weibull model *P(x)* built for Eclipse 2.0 as follows:

$$n = m/P(0.2N)*P(N)$$

where *N* is the total number of modules in Eclipse 3.0.

To evaluate the accuracy of the prediction method, we use the metric MRE, which is defined as $MRE = |N - N^{\hat{}}|/N$, where N and $N^{\hat{}}$ are the actual and estimated values, respectively. The value of MRE is between 0 and 1. The commonly acceptable value is 0.25 (the smaller the value the better the estimation).

Table 4 shows the prediction results for Eclipse versions 2.1 and 3.0 based on the Weibull model built for the Eclipse 2.0 and the defect data from the top 20% of the largest modules. The actual and predicted numbers of defects are reported. All MRE values are below 25%, falling within the acceptable levels. The average MRE values are 6.46% (package level) and 10.91% (file level). Similar results are obtained when we use the Weilbull model for Eclipse 2.1 to predict for Eclipse 3.0. These results show that we can predict the total number of defects effectively using the Weibull model built from historic data.

6. Predicting Defect-Prone Components

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 5. We explore the effectiveness of identifying defective components based on LOC.

Table 5. 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

6.1 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.

Table 6. The results of a prediction model

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

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 6.

In this project, we have chosen five commonly used classification techniques as shown in Table 7. 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 make 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 folds (segments) randomly. Each fold in turn is held as the test data and a classification model is trained on the rest nine folds.

Table 7. 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

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

6.2 Accuracy measures

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

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

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.

6.3 Validation results

Table 8 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 predication 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.

Table 8. Cross-Validation results of LOC-based classification model (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%

7. 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.

The NASA dataset is stored in the NASA IV&V Facility Metrics Data Program (MDP) repository, which is available for public[3]. It contain software measurement data and associated defect data that is collected from many NASA projects such as flight control, spacecraft instrument, storage management, and scientific data processing. In this study, we analyze 10 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 9).

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 rank order correlation shows that there is a weak but significant relationship between LOC and defect data (at the 0.01 level, 2-tailed).

Further analysis show that, when the modules are ordered by LOC, the distribution of defects follows the Weibull distribution. Table 10 shows the parameters of Weibull distributions, as well as the fitness measures. The R^2 values range from 0.948 to 0.998 and the S_e values range from 0.011 to 0.045. These results confirm our findings reported in Section 4.

As in the case of Eclipse, we can predict the total number of defects for new projects based on Weibull models constructed from the similar projects developed by the organization. In the replication study, we use the Weibull model built for the CM1 project to predict the total number of defects in other projects, once we have the defect data for the top 20% of the largest modules of these projects. The prediction results are shown in Table 11. All MRE values are within acceptable range (below 25%), except for the projects PC2 and MC1. For the project MC1, the MRE is 26.58%, which is just slightly above the acceptable criteria. For project PC2, the large 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 12). In Table 11, the average MRE value is only 14.28%. Therefore, we conclude that the replication results support the validity of our defect prediction method that is described in Section 5.

Table 9. The NASA dataset

Project	Total LOC	#Components	#Functions	# 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
MW1	8K	1	8341	37	C
Total	**265K**	**182**	**267986**	**1716**	

Table 10. The Weibull distribution of NASA defects

Project	γ	β	R^2	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
MW1	0.187	0.714	0.955	0.045

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

Table 11. Predicting the total number of defects based on Weibull model of the CM1 project

Project	Actual number of defects	Predicted number of defects	MRE	MRE <=25%?
PC1	139	150	7.91%	yes
PC2	26	38	46.15%	no
PC3	259	250	3.47%	yes
PC4	367	413	12.53%	yes
KC1	525	526	0.12%	yes
MC1	79	100	26.58%	no
MC2	113	108	4.42%	yes
KC3	101	115	13.86%	yes
MW1	37	42	13.51%	yes

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 analyzes the Product Hierarchy document and aggregates 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 12 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 6.

Table 12. 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

8. Related Work and Discussions

Work on predicting the number of defects based on LOC

Many researchers have studied the relationship between LOC and defects. Fenton and Ohlsson [5] plotted a diagram that displays the accumulated percentage of the number of defects when modules are ordered with respect to LOC for a telecommunication system. They concluded that LOC is quite good at ranking the top 20% of the defect-prone modules. They termed it the "ranking ability" of LOC. However, they did not elaborate their findings, nor provided any detailed statistical analysis. Ostrand et al. [12] studied the "ranking ability" of LOC for an inventory system and a provisioning system. They found that 20% of largest files contain 73% and 74% of the defects for the two systems. However, they considered the LOC-based model a "quick and dirty" method for ranking files, and suggested using a full negative binomial regression based model instead. The full model used many predictor variables including LOC, file change status, file age, defects found in previous release, the programming language and the release number.

In this paper, we perform a statistical analysis of the ranking ability of LOC. We discover the Weibull distribution of defects when modules are ranked by LOC. We have also proposed a method to predict the total number of defects based on Weibull models constructed from similar projects. The evaluations results show that the proposed method is simple yet effective.

Work on predicting defective modules based on LOC

Many researchers have constructed defect prediction models 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.

9. Conclusions

The use of static code attributes such as LOC for defect prediction has been widely debated. In this paper, we analyzed two public defect datasets: the Eclipse dataset and the NASA dataset. Our results confirm that LOC can be a useful indicator of software quality, and that we are able to build useful defect prediction models based on LOC. The contributions of this paper are as follows: Firstly, we discover the Weibull distribution of defects when modules are ranked with respect to LOC. Secondly, we show that we can use the Weibull models built from similar projects to successfully predict the total number of defects. Thirdly, using typical classification techniques, we are able to predict defective components reasonably well based on LOC.

Many researchers have performed quantitative analysis of defects and module sizes based on organizations' internal data [1, 2, 4, 5, 10, 12]. Their results cannot be verified and compared. By using the open Eclipse and NASA datasets, we hope that other researchers could repeat our experiments and improve our method.

In future, we will further evaluate the potential of our defect prediction method for a wide variety of large-scale software projects. We hope our work could help achieve a better understanding of software defects.

Acknowledgments

This research is supported by the 6th Key Researcher Program funded by the Tsinghua University, and the Chinese NSF grants 90718022 and 60703060.

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), pp. 340-355, 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/.