# Automated Identification of Change-Prone Classes in Open Source Software Projects

Xiaoyan Zhu, Qinbao Song, Zhongbin Sun
Xi'an Jiaotong University, Xi'an, Shaanxi, China 710049
Email: xyxyzh@gmail.com, qbsong@mail.xjtu.edu.cn, zhongbin@stu.xjtu.edu.cn

*Abstract*— **Identifying change-prone classes can enable developers to pay more attention to classes with similar characteristics in the future and thus test resources and time can be used more effectively. In this paper, we collect a set of static metrics and change data at class level from an open-source software product, Datacrow. With this data, we first validate Pareto's Law and find that about 80% of the lines changed are located in only 20% of the classes. We then use classification methods to identify these change-prone classes. Our experimental results show that our classification results are useful for identifying change-prone classes and thus can help to improve the efficiency of developers.**

*Index Terms*— **open-source software, change-prone classes, static metrics, classification methods.**

## I. Introduction

Open-source software is of increasing importance nowadays. More and more companies are investing in open-source software and profit from developing and maintaining it. As a result, software products are becoming very large and complex. Maintaining these software products requires a large amount of effort. But sometimes resources and time are limited and testing a software product in an exhaustive way is infeasible. In this case, using these limited resources and time effectively becomes an extremely important goal for developers to boost the competitiveness of their company. Identifying change-prone classes and then distributing more resources and time to these classes can help developers to achieve this goal.

To identify change-prone classes, we must first make sure that such classes exist. That is to say, we must make sure that some classes are more likely to change than others. Suppose that all the classes have the same probability to change, identifying change-prone classes would not work. For close-source software, Porter and Selby [1] stated with the 80:20 rule that approximately 20 percent of a software system is responsible for 80 percent of its faults, costs, and rework. This phenomenon is generally referred to as Pareto's Law.

Can Pareto's Law be applied to open-source software? Koru and Liu [2] validated Pareto's Law to two open-source projects and their experimental results strongly support Pareto's Law. But they used change count, which indicated the number of times a class was changed, to measure the change-proneness of a class. With the consideration that a change on one line and a change on 100 lines should not be treated equally in the measurement of the change-proneness of a class, in this paper, we used changed-line instead of change count as the label of a class to reflect its change proneness. In this case, can Pareto's Law still be applied? This is the first question to answer in this paper.

If the answer to the first question is yes, how can developers identify this small proportion of change-prone classes? This is the second question to answer in this paper.

To answer the two questions brought forward above, we conducted experiments with data extracted from an open-source software product, Datacrow. This data includes both static metrics data and changed-line data, which is used to reflect the change-proneness of a class. The data collection process will be described in detail later.

The remainder of the paper is arranged as follows. Section II introduces some related work. Section III is devoted to the description of research method. Section IV focuses on data collection. Section V validates Pareto's Law. The experimental results are reported in Section VI. Finally, the paper is concluded in Section VII.

## II. Related Work

Many researchers have studied the relationship between static metrics and risk factors, such as change, defects and effort [2] [3] [4] [5]. They have also tried to predict these factors with static metrics.

Some work paid attention to predicting maintenance effort using static metrics. Li and Henry [3] conducted research with data collected from two commercial software systems. They found a strong relationship between the metrics and the maintenance effort, which is measured by the number of lines changed per class. They also found that the maintenance effort can be predicted from the combination of metrics. Bayesian network [6] and multivariate adaptive regression splines [7] were further used to build maintenance effort prediction models with the data collected by Li and Henry [3].

Some work was concerned about fault-proneness. Gyimothy et al. [4] calculated the object-oriented metrics given by Chidamber and Kemerer [8]. They employed both well-known statistical methods and machine learning techniques to predict fault-proneness for open-source software. Vandecruys et al. [5] used software metrics to classify software modules as fault-prone or not fault prone. They used a classification technique AntMiner+

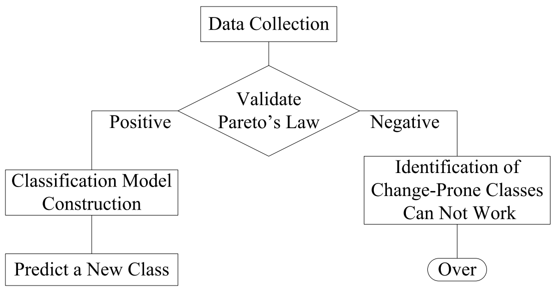
Data Collection
Validate Pareto's Law
Positive
Negative
Classification Model Construction
Identification of Change-Prone Classes Can Not Work
Predict a New Class
Over

Figure 1. Analysis Pipeline.

to predict erroneous software modules. Kim et al. [9] used a machine learning classifier to predict bugs in file-level software changes. Their classifier aimed to explore whether there was a bug in any of the lines that were changed in one file in one SCM commit transaction. Information features and source code terms are both used in the building of prediction model. Other work concentrated on predicting change-proneness, which is most related to our work. Koru and Liu [2] collected a set of static metrics and change data at class level from two open-source projects. Using this data, they found that a great majority of changes are rooted in a small proportion of classes. They then identified and characterized the change-prone classes by producing tree-based models.

Our work focuses on identifying change-prone classes in open-source software. This is different from works that used data from closed-source software [3] [6] [7] and works that focused on predicting fault-proneness [4] [5] [9]. We use changed-line to reflect the change-proneness of a class. This is different from work [2] that used change count instead to reflect the change-proneness of a class.

## III. Research Method

In this section, we present a big picture overview of the research method. Our research method consists of three parts. An outline is shown in Figure 1.

As seen in Figure 1, we start by first collecting the necessary data. We collect this data at the class level and treat each class as a data point. Each data point includes a set of static metrics and a changed-line label, which reflects the change-proneness of the corresponding class. To validate Pareto's Law, we first sort all of the classes according to their changed-line labels in descending order. We then calculate the cumulative sum of the changed-line labels. With this data, we can determine the dis- tribution of the changed lines across classes and get the answer to the first question. If 80% of the changed lines are not concentrated in 20% of the classes, Pareto's Law does not hold and the identification of change-prone classes using our method can not work here. If Pareto's Law does hold, we can build a classification model to identify change-prone classes. To obtain a training data set, we substitute the changed-line label with a change-proneness label, which states whether a class is change-prone. The top 20% of classes by the number of changed lines are labeled as change-prone and the remaining 80% are labeled as not-change-prone. A classification model can be built using this new set of data points. Such a model is called a simple model.

Some classes, such as very small classes, library classes or reused classes may have no line changed. Their features may be different from other not change-prone classes. Thus, classes that are not change-prone should be further divided into two categories. Classes with no changed lines are labeled as no-change and the other not change-prone classes are labeled as change. A classification model can be built using the set of data points with new labels. Such a model is called an improved model. With a classification model, developers can predict whether a class is change-prone and thus can decide whether more attention should be paid to it.

In the following sections, we collect data from Datacrow and conduct experiments to test our method using the collected data.

## IV. Data Collection

We analyzed the static metrics and changed-line data from Datacrow 3_4_0, an open-source product written in Java, in this paper. Release 3_4_0 was chosen for that it was the first major release in the history log. Static metrics were obtained using Understand [1] and Software-Metrics-in-Eclipse [2]. In all, 68 metrics, 41 from Understand and 27 from Software-Metrics-in-Eclipse, were extracted for each data point.

Changed-line label for a class was calculated from the history log, which can be obtained from CVS (Concurrent Versions System). Changed-line is the sum of all the added and deleted lines of revisions from the release date for Datacrow 3_4_0 to the data extraction date. Therefore, changed-line for a given class c in Datacrow 3_4_0 from the history log can be calculated in the following procedure: 1) Firstly, we check if release 3_4_0 of Datacrow has any revisions for class c. If the answer is negative, we can omit it because no metrics data can be extracted for c in Datacrow 3_4_0; 2) If the answer is positive, we find revision r of c corresponding to release 3_4_0; 3) The sum of all the added lines and deleted lines for revisions from r to the latest one is the changed-line for c.

At last, 490 data points were obtained. Each point has 68 static metrics data as its features and a changed-line label.

## V. Validating Pareto's Law

In this section, we introduce the results regarding the validity of Pareto's Law for Datacrow.

Using the cumulative sum data of changed-line of Datacrow, we drew a plot as shown in Figure 2. Sorted data points are aligned on the x-axis from left to right. The tick marks on the x-axis show the percentile of classes and those on the y-axis show the percentile of cumulative sum of changed lines. The horizontal dashed line shows

[1] http://www.scitools.com/
[2] http://agile.csc.ncsu.edu/SEMaterials/tutorials/metrics/

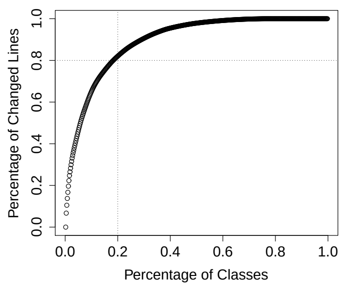

Figure 2. Validating Pareto's Law for Datacrow.

the 80th percentile of cumulative sum of changed lines and the vertical dashed line shows the 20th percentile of classes.

From Figure 2, we can see that the intersection of the horizontal dashed line and the vertical dashed line almost meets the curve. This reveals that about 80% of lines changed are located in about 20% of classes, providing strong support for Pareto's Law. We can conclude from the result that a small proportion (20%) of classes are more likely to be changed than a large majority (80%) of other classes. Identifying this small proportion of classes could help developers to improve their efficiency.

## VI. Identification Results and Analysis

From the discussion in Section 5, we can see that the top 20% of classes, or about 98 classes, in Datacrow are change-prone. In this section, we show and analyze the results of the identification of change-prone classes using data from Datacrow.

### A. *Data set*

To test the classification results of the simple and improved models, we conducted an experiment using data collected from Datacrow. In the data set for the simple model, 98 classes have change-prone label and 392 classes have not change-prone label. In the data set for the improved model, 98 classes have change-prone label, 275 classes have change label and 117 classes have no-change label. When the improved model is used, the predicted label of a new class is not change-prone if it is predicted as change or no-change. So the classification result is correct for a not change-prone class to be predicted as either change or no-change.

### B. *Setup*

To make full use of the data set and obtain more realistic estimates, we used 10-fold cross-validation in the experiment. For each fold, 9/10th of the data set was used as the training set to build a classification model. The remaining 1/10th of the data set was used as the test set. We used five different kinds of classification methods, including a statistical method, Naive Bayes (NB) [10]; a tree-based method, C4.5 [11]; an instance-based classification method, k -NN [12]; a kernel-based method, SVM [13]; and an associative classification method, ACWV [14].

### C. *Evaluating measures*

Accuracy is an important measure for the evaluation of classification results. In our data set, class labels are unevenly distributed. Not change-prone data points are much more numerous than change-prone data points. Moreover, we aim to identify as many change-prone classes as possible. As such, we should consider the recall of the classification results in addition to the accuracy.

### D. *Classification results*

Table I shows the results of different classification methods for both the simple and improved models.

From Table I, we can see that when the random method, or no classification method, is used, results obtained from the simple and improved models are identical. For the random method, accuracy and recall-2 are 80.0%, and recall-1 is 20.0%. The accuracy and the recall-2 are acceptable due to uneven distribution of change-proneness labels in the data set. However, the recall-1 is so low that only 19 out of 98 change-prone classes are identified, which is not acceptable. Thus, identification methods are required to identify more change-prone classes. We analyze classification results of the simple and improved models separately in the following.

We first analyzed the results when the simple model was used. The accuracies of all of the classification methods are higher than 80.0% and are all acceptable. SVM obtained highest accuracy among all the classification methods. However, SVM identified only 30.6% of change-prone classes. Considering that our aim is to identify change-prone classes, we should place more importance on their correct classification. In this way, ACWV is better than the other classification methods. What is more, its accuracy of 82.2% is as good as those of the other methods.

We then analyzed the results when the improved model was used. For NB, SVM and ACWV, accuracy was improved over that obtained when the simple model was used while for C4.5 and k-NN, accuracy decreased. SVM still obtained highest accuracy among all the classification methods. However, its recall-1 is only 35.7%, which is not acceptable. ACWV is still the best method considering both accuracy and recall-1. No matter which model, the simple model or the improved model, is used, ACWV obtained better result than the other methods. We can conclude that ACWV is more appropriate for the identification of change-prone classes for Datacrow than the other methods.

When using ACWV in improved model, 61.2% of the change-prone classes and 88.3% of the not change-prone classes are correctly classified. This means that 61.2% of the change-prone classes can be identified and paid more attention to by developers. Such ability is extremely useful when resources and time are precious. Suppose that only 20% of the classes can be tested due to resource and time limitations. In this situation, if developers select 20% of the classes randomly, only 20% of the change-prone classes will be tested. Thus, only a

TABLE I.
CLASSIFICATION RESULTS(%)

| Classification Method | Simple Model | | | Improved Model | | |
|---|---|---|---|---|---|---|
| | Accuracy | Recall-1 | Recall-2 | Accuracy | Recall-1 | Recall-2 |
| Random | 80.0 | 20.0 | 80.0 | 80.0 | 20.0 | 80.0 |
| NB | 82.9 | 51.0 | 90.8 | 83.7 | 45.9 | 93.1 |
| C4.5 | 82.7 | 50.0 | 90.8 | 79.6 | 49.0 | 87.2 |
| k-NN(k=3) | 81.0 | 36.7 | 92.1 | 77.3 | 42.8 | 86.0 |
| SVM | 84.3 | 30.6 | 97.7 | 85.3 | 35.7 | 97.7 |
| ACWV | 82.2 | 60.2 | 87.8 | 83.1 | 61.2 | 88.3 |

small percentage of test resources and time can be used on change-prone classes. But with the classification results, most (more than 60%) test resources and time can be used on change-prone classes, which is a strong support to the validity of the identification of change-prone classes using classification methods.

### E. Analysis of incorrectly classified classes

Regardless of the classification method used, some change-prone and not change-prone classes are not correctly classified. Incorrectly classified classes will misdirect the distribution of test resources and time. As such, we have to further analyze these classes, including classes incorrectly classified as change-prone and not change-prone. As discussed in Subsection 6.4, ACWV obtained better results than the other classification methods. Therefore, here we only analyze the results using ACWV. In Section 5, to validate Pareto's Law, we sorted all of the data points according to their changed-line label in descending order. Here, to facilitate analysis, we give each data point an ID that indicates its position in the ordered data set. Six statistics, which are count, mean, median, min, max and standard deviation (STD), are used to describe the IDs of incorrectly classified classes. Table II shows the values of these statistics.

From Table II, we can see that the improved model produces better result than the simple model does on the whole. The number of incorrectly classified change-prone classes and not change-prone classes found using the improved model are both less than those obtained using the simple model. The min ID of incorrectly classified change-prone classes found using the improved model is higher than that obtained using the simple model. The mean value of incorrectly classified not change-prone classes found using the improved model is higher than that obtained using the simple model. The other values for the two models are the same or similar.

Incorrectly classified change-prone classes will not be paid much attention to during testing as they are classified as not change-prone. If most of these classes have a large amount of changed lines, they will increase the maintenance effort. Figure 3 shows the box plots of the IDs of these classes.

For a box plot, the bottom and top of the box are always the 25th and 75th percentile (the lower and upper quartiles, respectively), and the band near the middle of the box is always the 50th percentile (the median). The difference between the upper and lower quartiles is called the IQ. The line above the box identifies the value L1: upper quartile + 1.5*IQ. The line below the box identifies the value L2: lower quartile - 1.5*IQ. Points greater than L1 or less than L2 are outliers and are shown as black dots in the plot.

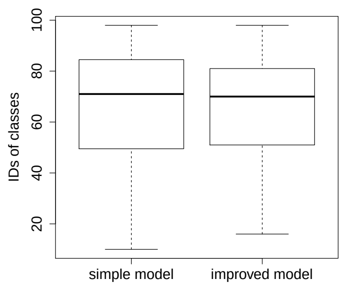

Figure 3. The IDs of incorrectly classified change-prone classes.

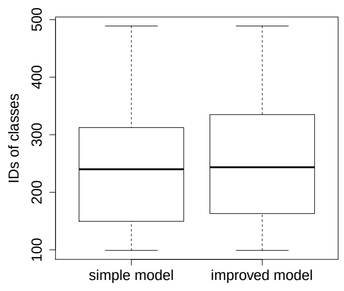

Figure 4. The IDs of incorrectly classified not change-prone classes.

From Figure 3, we can see that for the simple model, the lower quartile and upper quartiles are about 50 and 85, respectively. This means that most IDs range from about 50 to 85. We can also see that no IDs are below 10. This absence means that the most largely changed classes are all identified and can be tested thoroughly. For the improved model, most IDs range from about 50 to 82. No IDs are below 15, which is better than that of the simple model. We can conclude that whichever model is used, we have not omitted many largely changed classes.

Not change-prone classes that are incorrectly classified as change-prone will take up test resources and time, a lot of which will be wasted if most of these classes have only a few lines changed. Figure 4 shows the box plots of the IDs of these classes.

From Figure 4, we can see that most IDs are in the range of about 150 to 310 for the simple model and about 160 to 330 for the improved model. The upper and lower quartiles of the simple model are both less than that of the improved model, which indicates that simple model is

TABLE II.
STATISTICS FOR IDS OF INCORRECTLY CLASSIFIED CLASSES

| Statistics | Simple Model | | Improved Model | |
|---|---|---|---|---|
| | Change-prone | Not Change-prone | Change-prone | Not Change-prone |
| Count | 39 | 48 | 37 | 46 |
| Mean | 66 | 246 | 66 | 256 |
| Median | 71 | 240 | 70 | 244 |
| Min | 10 | 99 | 16 | 99 |
| Max | 98 | 489 | 98 | 489 |
| STD | 23 | 104 | 22 | 103 |

superior to improved model. This result is opposite to our conclusion obtained from Table I that improved model is better than the simple model. This may be due to the fact that simple model incorrectly classified two more classes than the improved model. The IDs of these two classes are small ones, thus lowering both the upper and the lower quartiles of the simple model.

We can also see that, for both the simple and improved models, only four IDs are below 374. In the ordered data set, classes whose IDs are below 374 have no line changed and totally there are 117 such classes. Therefore, using a classification model, only four of the 117 classes with no changed lines were classified as change-prone and would be tested. Without the classification model, 23 such classes will be tested when 20% of all classes are selected randomly to test. What is more, some IDs are very near 100, meaning that the corresponding classes are very close to the top 20%. We can conclude that although we will test some not change-prone classes, we will not waste a significant amount of test resources and time doing so. We find from the discussion above that, although some classes are incorrectly classified, the classification results are acceptable. With these results, test resources and time can be used much more effectively.

## VII. CONCLUSIONS

In this paper, we presented an empirical study that sought to identify change-prone classes. First, we collected static metrics data and change data from Datacrow. We then validated Pareto's Law and found that 80% of lines changed are located in about 20% of classes. Finally, classification methods were used to identify change-prone classes. The experimental results revealed that the classification results allow test resources and time to be used much more effectively.

## ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under grant 61070006.

We thank Bradley Collins from Stanford University for numerous discussions concerning this work. We thank Chuan Ding from Xi'an Jiaotong University for his assistance on the experiments.

## REFERENCES

[1] A. Porter and Richard W. Selby, "Evaluating techniques for generating metric-based classification trees," *Journal of Systems and Software*, vol. 12, no. 3, pp. 209–218, Jul. 1990.

[2] A. GUNESKORU and H. LIU, "Identifying and characterizing change-prone classes in two large-scale open-source products," *Journal of Systems and Software*, vol. 80, no. 1, pp. 63–73, Jan. 2007.

[3] H. S. Li W., "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, Nov. 1993.

[4] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, Oct. 2005.

[5] O. VANDECRUYS, D. MARTENS, B. BAESENS, C. MUES, M. DEBACKER, and R. HAESEN, "Mining software repositories for comprehensible software fault prediction models," *Journal of Systems and Software*, vol. 81, no. 5, pp. 823–839, May 2008.

[6] C. van Koten and A. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," *Information and Software Technology*, vol. 48, no. 1, pp. 59–67, Jan. 2006.

[7] Y. ZHOU and H. LEUNG, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1349–1361, Aug. 2007.

[8] S. Chidamber and C. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[9] Sunghun Kim, Whitehead, E.J., Y. Zhang. ,"Classifying Software Changes: Clean or Buggy?" *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 181–196, Mar. 2008.

[10] R. Duda and P. Hart, *Pattern classification and scene analysis*. Wiley, Aug. 1973.

[11] J. R. Quinlan, "C4.5: programs for machine learning," Mar. 1993.

[12] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.

[13] T. Joachims, "Text Categorization with Support Vector Machines: Learning with Many Relevant Features," Chemnitz,Germany, pp. 137–142, Apr. 1998.

[14] X. Zhu, Q. Song, and Z. Jia, "A Weighted Voting-Based Associative Classification Algorithm," *The Computer Journal*, vol. 53, no. 6, pp. 786–801, Aug. 2009.

**Xiaoyan Zhu** received her MS degree in computer science and technology from Xi'an Jiaotong University, China, in 2008.

She is currently a Ph.D. candidate at Xi'an Jiaotong University, Xi'an, China. Her research interests include software engineering and data mining.

**Qinbao Song** received the Ph.D. degree in computer science from Xi'an Jiaotong University, China, in 2001.

He is currently a Professor of software technology in the Department of Computer Science and Technology, Xi'an Jiaotong University, where he is also the Deputy Director of the

Department of Computer Science and Technology. His research interests include data mining, machine learning and software engineering.

**Zhongbin Sun** received his BS degree in computer science and technology from Xi'an Jiaotong University, China, in 2010.

He is currently a Ph.D. candidate at Xi'an Jiaotong University, Xi'an, China. His research interests include software engineering and data mining.