Evaluating Software Metrics for Sorting Software Modules in Order of Defect Count

Xiaoxing Yang[a]
School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China

Keywords: Software Defect Prediction, Sorting Modules in Order of Defect Count, Software Metrics, Correlation Analysis, Metric Analysis.

Abstract: Sorting software modules in order of defect count can help testers to focus on software modules with more defects. Many approaches have been proposed to accomplish this. In order to compare approaches more fairly, researchers have provided publicly available data sets. In this paper, we provide a new metric selection approach and evaluate the usefulness of software metrics of eleven publicly available data sets, in order to investigate the quality of these data sets and find out the software metrics that are most efficient for sorting modules in order of defect count. Unexpectedly, experimental results show that only one metric can work well over most of these data sets, which implies that more effective metrics should be introduced. We also obtain other findings from these data sets, which can help to introduce new metrics for sorting software modules in order of defect count to some extent.

1 INTRODUCTION

Software testing activities play a key role in software development, which consume a great amount of resources including time, money and personnel (Ostrand et al., 2005). Sorting software modules (files, packages, etc) in order of defect count can help testers to focus on software modules with more defects and identify defects more quickly (Nam and Kim, 2015). The process mainly includes two parts: data and model construction methods, and a simple example can be found in Yang et al.'s work (Yang et al., 2015).

Many approaches, such as negative binomial regression and random forest (Weyuker et al., 2010), have been proposed to construct prediction models. In order to compare approaches more fairly, researchers have provided publicly available data sets(D'Ambros et al., 2011; Zimmermann et al., 2007). These data sets include numerous software metrics, some of which have about 200 metrics. Existing software metrics include metrics based on software scale (Akiyama, 1971), metrics based on software complexity (McCabe, 1976), object-oriented metrics (Basili et al., 1996), and process metrics (Moser et al., 2008).

The inclusion of irrelevant, redundant, and noisy attributes can result in poor predictive performance and increased computation (Hall and Holmes, 2003). Therefore, some researchers study the effectiveness of software metrics to construct software defect prediction (SDP) models(D'Ambros et al., 2011; Rahman and Devanbu, 2013), and some researchers propose metric selection approaches in the domain of SDP, in order to solve problems such as high dimensionality problems (Khoshgoftaar et al., 2014; Liu et al., 2014). Many interesting findings have been found (Menzies et al., 2007; Rahman and Devanbu, 2013) and numerous useful metric selection methods have been proposed (Wang et al., 2011b). However, most of these studies are about software metrics to classify software modules into defect-prone and defect-free categories (Khoshgoftaar et al., 2014; Shivaji et al., 2013). In this paper, we analyze eleven publicly available data sets that include the number of defects in each software module (not only defect-prone or not), and evaluate the usefulness of their metrics to sort software modules according to the number of defects, in order to investigate the quality of these data sets and find out the software metrics that are most efficient for sorting modules in order of defect count.

In this paper, the key contributions include:

1. A new selection approach which can select efficient software metrics for sorting software modules according to the number of defects.

[a] https://orcid.org/0000-0001-8569-2832

Yang, X.
Evaluating Software Metrics for Sorting Software Modules in Order of Defect Count.
DOI: 10.5220/0007924800940105
In *Proceedings of the 14th International Conference on Software Technologies (ICSOFT 2019)*, pages 94-105
ISBN: 978-989-758-379-7

2. A comprehensive evaluation of software metrics to sort software modules in order of defect count over eleven publicly available data sets, including six Eclipse data sets (file-level and package-level) (Zimmermann et al., 2007) and five other data sets (D'Ambros et al., 2011).

The rest of this paper is organized as follows: Section 2 presents related work. In Section 3, we detail our selection approach. Experimental setup is described in Section 4 and experimental results are reported in Section 5. Threats to validity are discussed in Section 6. Section 7 draws the conclusions.

2 RELATED WORK

Sorting software modules in order of defect count is one kind of software defect prediction (SDP), which is also called as SDP for the ranking task in this paper. It employs software metrics to predict their defect information in order to support software testing activities (D'Ambros et al., 2011; Yang et al., 2015). As mentioned above, data and model construction methods are key factors for SDP for the ranking task. Since we focus on the effectiveness of software metrics in this paper, we review the related work mainly from data aspect.

In the beginning, size and complexity metrics, such as lines of code(Akiyama, 1971) and Halstead's metrics (Halstead, 1997), were widely used in an attempt to predict the number of defects that might reveal in operation or testing(Fenton and Neil, 1999).

With the development of software, more and more software metrics were introduced in predicting defect-proneness. For example, Ohlsson and Alberg (Ohlsson and Alberg, 1996) used design metrics, which were derived automatically from design documents, to research on the identification of fault-prone modules. Childamber and Kemerer (Chidamber and Kemerer, 1994) developed and implemented a new set of software metrics for object oriented design, and presented empirical data on these metrics from actual commercial systems, which demonstrated the usefulness of these metrics. Graves et al.(Graves et al., 2000) explored the extent to which measurements from the change history were successful in predicting the distribution of these incidences of faults, and they found that process measures based on the change history were more useful in predicting fault rates than product metrics of the code.

The inclusion of irrelevant, redundant, and noisy attributes can result in poor predictive performance and increased computation (Hall and Holmes, 2003). Therefore, the rationality and effectiveness of these metrics have been questioned and analyzed. For example, Churcher and Shepperd argued that a number of fundamental issues should be clarified about the object oriented metrics developed by Chidamber and Kemerer before applying them (Churcher and Shepperd, 1995). A lot of work has been done to analyze the effectiveness of software metrics, and we divide them into three categories.

The first category is direct comparison. That is, software metrics are categorized into different sets, and then the effectiveness of these different sets of metrics is compared by comparing the performance of models constructed by corresponding metrics. D'Ambros et al. (D'Ambros et al., 2011) compared many sets of metrics over five data sets. Experimental results showed that process metrics, churn of source code metrics and entropy of source code metrics were more effective than other metrics for building SDP models. Graves et al. (Graves et al., 2000) analyzed metrics for predicting the distribution of defects, and concluded that process metrics were more useful than product metrics. Moser et al. (Moser et al., 2008) compared models respectively based on product metrics and process metrics, and concluded that process metrics were more useful than product metrics for the Eclipse data. Rahman and Devanbu (Rahman and Devanbu, 2013) analyzed the applicability and efficacy of process and code metrics from several different perspectives, and they found that code metrics were generally less useful than process metrics for prediction. This kind of methods can conclude which set of metrics are more effective but cannot tell which specific metric plays an important role.

The second category is correlation coefficients. Ohlsson et al.(Ohlsson and Alberg, 1996) used correlation coefficients to investigate the relationship between metrics and number of defects, and the relationship among eleven metrics. They found that the metrics that most correlated with the number of defects were highly correlated with each other. Graves et al. (Graves et al., 2000) computed correlation coefficients among complexity metrics and found that most complexity metrics were highly correlated to lines of code, which implied the existence of redundant metrics. Zimmermann et al.(Zimmermann et al., 2007) applied Spearman correlation coefficients (SCC) to analyze the relationship between complexity metrics and number of defects over Eclipse3.0, and found that most SCC were small. Hence, they pointed out that it was unlikely to used one metric to predict the number of defects. This kind of methods can reflect the relevance between metrics and number of defects in some degree, but may not reflect the effectiveness of software metrics for building models, which will be

discussed in Section 3.

The third category is metric selection. Researchers employed feature selection methods from data mining domain to rank metrics or select a subset of metrics. Menzies et al. (Menzies et al., 2007) used Information Gain (IG) to select metrics. Experimental results showed that two or three metrics could work as well as all metrics, but models based on only one metric performed inferiorly. They presented the specific selected metrics, which included loc_blanks and call_pairs. Khoshgoftaar et al. (Khoshgoftaar et al., 2012) compared seven metric selection methods and found that IG and signal to noise ratio were better than other methods for selecting useful metrics to construct SDP models. Wang et al. (Wang et al., 2011a) applied five metric selection methods to SDP and pointed out that three metrics were enough to construct an effective SDP models, some of which were even better than models constructed based on all metrics. Wang et al. (Wang et al., 2011b) proposed an ensemble method and showed its effectiveness by comparing it with six other metric selection methods for SDP. This kind of methods can reflect the effectiveness of metrics for building SDP models, although most existing studies are about classifying software modules into defect-prone and defect-free categories instead of sorting modules in order of defect count.

According to Catal et al.'s review (Catal and Diri, 2009) and Jabangwe et al.'s review (Jabangwe et al., 2015), many existing studies used private data sets, and they pointed out the need to use publicly available data sets to enable the research community to validate and compare each others findings. One of the most popular used publicly data sets are the NASA data sets, and some researchers have investigated their quality(Gray et al., 2011; Shepperd et al., 2013).

Most above-mentioned related studies were about metrics to classify software modules. Therefore, there is a comparative lack of investigation of publicly data sets for SDP for the ranking task. Yang et al. found that two or three metrics worked well for SDP for the ranking task over some data sets by applying IG to select partial software metrics (Yang et al., 2015). In this paper, we follow their work to further investigate the effectiveness of software metrics for sorting modules in order of defect count over six Eclipse and five other publicly available data sets.

3 OBJECT-BASED METRIC SELECTION METHOD

In this section, we first explain why the second category (correlation coefficients) mentioned in Section 2 may not reflect the effectiveness of software metrics for sorting modules in order of defect count. Subsequently, we present our object-based metric selection (OMS) method.

As mentioned in Section 2, correlation coefficients are commonly used for investigating the relationship between metrics and number of defects (Zimmermann et al., 2007), or evaluating the performance of SDP models (D'Ambros et al., 2010). However, Spearman correlation coefficients (SCC) and Pearson correlation coefficient (PCC) might be inappropriate for evaluating models for sorting modules in order of defect count. In Table 1, we give a simple example to explain this. According to PCC, model 3 is better than model 2. However, model 2 and model 3 give the same order of modules. They would lead to the same allocation of testing resources. According to SCC, model 1 and model 2 are the same. However, when there are limited testing resources, only modules with largest predicted values will be tested, so the ranking of modules with more defects is more important. To be specific, when the limited testing resources can only test one module, model 2 would choose module D with twelve defects and model 1 would choose module C with only three defects. Hence, model 2 is better than model 1 for sorting modules in order of defect count. Therefore, both SCC and PCC are not suitable for evaluating models for sorting modules in order of defect count. Similarly, metrics most correlated with number of defects according to correlation coefficients might not be most effective for sorting modules according to number of defects.

Table 1: An Example: Insufficiencies of Correlation Coefficients.

Module Name	A	B	C	D
the actual defect number	1	2	3	12
predicted by model 1	1	2	3	2
predicted by model 2	3	2	3	5
predicted by model 3	3	2	3	15

If a metric can lead to a most desired ranking model, the metric is most effective for sorting modules in order of defect count. Therefore, we can evaluate the effectiveness of a metric by directly computing performance of the model based on the metric, which is the main idea of our OMS method.

OMS is directly based on the model performance, so we first describe the performance measure for sorting modules in order of defect count. In this paper, we adopt fault-percentile-average(FPA) (Weyuker et al., 2010) as the performance measure because FPA takes into account both practical use and the whole ranking performance of prediction models, which is demon-

strated to be consistent with Alberg diagram (Ohlsson and Alberg, 1996) (also referred to as cumulative lift chart) for measuring a ranking (Yang et al., 2015). Considering k modules $f_1, f_2, \ldots, f_k$, listed in increasing order of predicted defect number, n_i as the actual defect number in the module f_i, and n=n_1+n_2+...+n_k as the total number of defects in all modules, the proportion of actual defects in the top m predicted modules to the whole defects is $\frac{1}{n}\sum_{i=k-m+1}^{k} n_i$. Then FPA is defined as follows(Weyuker et al., 2010):

$$\frac{1}{k}\sum_{m=1}^{k}\frac{1}{n}\sum_{i=k-m+1}^{k} n_i$$

Actually, FPA is an average of the proportions of actual defects in the top i(i: *1* to k) predicted modules. Larger FPA means better ranking performance.

Using FPA as the model performance measure, the effectiveness of one metric can be evaluated by directly computing the FPA value of the model based on the metric, which is detailed in Algorithm 1.

Algorithm 1: Metric Analysis based on FPA.

input : a set of metrics $A = (A_1, A_2, \cdots, A_d)$
for $i = 1$ **to** d **do**
  to use A_i to predict the number of defects and compute the corresponding FPA_i
end
to rank metrics according to FPA values
return *an order of metrics*

Metric analysis based on FPA (Algorithm 1) directly use FPA of models based on each metric, so it can be used to analyze the effectiveness of each metric for sorting modules in order of defect count. However, when metrics are highly correlated with each other, Algorithm 1 is not suitable for selecting a subset of (more than one) metrics for building models, because the selected metrics might be highly correlated with each other (in another words, there exist redundancy among the selected metrics). Therefore, we propose the OMS method, which uses both FPA values and SCC to measure the effectiveness of selected metrics. Details are given in Algorithm 2.

4 EXPERIMENTAL SETUP

In this section, we detail research questions, data sets, and implementation respectively.

Algorithm 2: Object-based metric selection (OMS).

input : a set of metrics $A = (A_1, A_2, \cdots, A_d)$
  selected number: selectNum
  subset S (initialized as empty)
for $i = 1$ **to** d **do**
  to use A_i to predict the number of defects and compute the corresponding FPA_i
end
for $j = 1$ **to** *selectNum* **do**
  for $i = 1$ **to** d **do**
    if A_i *does not belong to* S **then**
      to compute the effectiveness of A_i

$$E_i = FPA_i - \frac{|\sum_{j \varepsilon S} SCC_{i,j}|}{|S|} \quad (1)$$

      where $SCC_{i,j}$ is SCC between A_i and A_j
    end
  end
  A_{max} is the metric with maximum effectiveness E outside S, then S_j=A_{max}
  $j++$
end
return *subset of selected metrics* S

4.1 Research Questions

The main research questions are as follows:

* **RQ1**: How many software metrics are most appropriate and which metrics are most effective for sorting software modules in order of defect count over eleven data sets?
* **RQ2**: Is there a small subset of metrics that can lead to good models for sorting modules in order of defect count over all data sets?

RQ1 could evaluate metrics for different data sets, while RQ2 could evaluate whether partial metrics can capture all necessary information over all data sets, or different partial metrics are needed for different data sets. Before investigating the two research questions, we need to answer the following questions:

* **Q01**: Are the best software metrics for SDP for the ranking task different from those for SDP for the classification task?
* **Q02**: Is our OMS method comparable to Information Gain (IG) for selecting partial metrics for sorting software modules in order of defect count?

The reason to investigate Q01 firstly is that we may directly employ the relevant results from SDP for the

classification task if the answer for Q01 is a 'No'. Even though our OMS method is directly based on the model performance and should be able to select effective metrics, if the answer for Q02 is a 'No', there is no need to apply OMS method to investigate metrics. That's why we should investigate Q02.

4.2 Datasets

In order to facilitate others to reproduce results, we use eleven publicly available data sets, including five data sets proposed by D'Ambros et al. (D'Ambros et al., 2011) and six Eclipse data sets proposed by Zimmermann et al. (Zimmermann et al., 2007). The benchmark[1] presented by D'Ambros et al. includes data over a five year period from five open-source software systems—Eclipse JDT Core (eclipse), Eclipse PDE UI (pde), Equinox framework (equinox), Mylyn, and Apache Lucene (lucene). There are several sets of metrics for these five systems. We combine all metrics together, and hence we can get five datasets with one release. The Eclipse datasets provided by Zimmermann et al. have three releases for both file-level and package-level, so SDP models could be used for predicting defects of a new release, which is more like the actual situation. For avoiding misunderstandings, we denote the latter Eclipse as Eclipse_II. The characteristics of the experimental data sets are shown in Table 2. The column of 'faulty modules' records the number of modules having defects with the percentages of faulty modules in the subsequent brackets, the column of 'range of defects' records ranges of defect numbers, and the column of 'total defects' records the total number of defects in all modules of the corresponding data sets.

Table 2: Experimental Data Sets.

Datasets name	module number	metric number	faulty modules	rangeof defects	total defects
Eclipse_II_File2.0(File2.0)	6729	198	975(14.5%)	[0,31]	1692
Eclipse_II_File2.1(File2.1)	7888	198	854(10.8%)	[0,9]	1182
Eclipse_II_File3.0(File3.0)	10593	198	1568(14.8%)	[0,17]	2679
Eclipse_II_Package2.0 (Package2.0)	377	207	190(50.4%)	[0,88]	917
Eclipse_II_Package2.1 (Package2.1)	434	207	194(44.7%)	[0,71]	662
Eclipse_II_Package3.0 (Package3.0)	661	207	313(47.4%)	[0,65]	1534
eclipse	997	212	206(20.7%)	[0,9]	374
equinox	324	212	129(39.8%)	[0,13]	244
lucene	691	212	64(9.3%)	[0,9]	97
mylyn	1862	212	245(13.2%)	[0,12]	340
pde	1497	212	209(14.0%)	[0,28]	341

[1] http://bug.inf.usi.ch/

4.3 Implementation

In this subsection, we present the implementation approaches.

According to the research questions, we conduct the following experiments.

1. In order to answer Q01, we use IG as the selection method, since it has been applied to both classifying modules (Menzies et al., 2007) and sorting modules(Yang et al., 2015). To be specific, we apply IG to select three metrics according to both ranking (sorting modules) and classification tasks, and compare the results, in order to see whether the selected metrics are different. If there exist differences among the selected metrics, the answer is a 'Yes'. Otherwise the answer might be a 'No', and we should dig deeper to find the answer. Since the dependent variable is number of defects for sorting modules, we convert the dependent variable by defining zero as defect-free and other values as defect-prone for the classification task.
2. We use Yang et al.'s strategy (Yang et al., 2015) to conduct experiments to answer Q02. To be specific, we compare OMS with IG by comparing the performance of models based on different numbers of selected metrics over eleven data sets, using the learning-to-rank (LTR) as model construction method.
3. In order to investigate RQ1, we use OMS or IG (depending on the comparison results of Q02) as the selection method, and use random forest(RF) as the model constructed method, which was demonstrated to perform best over the original data sets (Yang et al., 2015). In order to investigate which metrics are most effective for sorting modules in order of defect count, we use Algorithm 1 to rank software metrics over all data sets (the metric with largest FPA values ranks 1, so the metric with smallest rank value is most effective).
4. For RQ2, we use a small subset of most effective metrics to construct prediction models to see whether partial metrics can capture all necessary information for sorting modules over all data sets. Since data sets have different original metrics, we categorize data sets according to their metrics firstly, and use different subsets of metrics for different categories. A 'Yes' answer for RQ2 might imply that other metrics should not be used because they are redundant or noisy for sorting modules in order of defect count, and A 'No' answer might imply that we still should keep other metrics because the redundant or noisy metrics for

Table 3: Three Best Metrics Selected by IG According to Different Tasks and Mean FPA of Their Corresponding Models.

datasets name	classification results	ranking results	selected metrics according to classification	selected metrics according to ranking
Files2.0	0.801	0.804	NBD_max,SimpleName NORM_VariableDeclarationStatement	NBD_max,SimpleName SUM
Files2.1	0.765	0.765	NBD_max,SimpleName,SUM	SimpleName,NBD_max,SUM
Files3.0	0.737	**0.787**	NBD_max,NORM_MethodInvocation NORM_TypeDeclaration	NBD_max,Block SimpleName
Package2.0	0.769	0.766	NBD_max,PAR_max ImportDeclaration	NBD_max,NOM_max ImportDeclaration
Package2.1	0.759	**0.774**	NBD_max,PAR_max ImportDeclaration	NBD_max,Modifier ImportDeclaration
Package3.0	0.811	0.807	NBD_max,TLOC_max ImportDeclaration	NBD_max,NOCU TLOC_max
eclipse	0.824	0.824	CvsEntropy,ent-wmc ent-numberOfLinesOfCode	CvsEntropy,ent-wmc ent-numberOfLinesOfCode
equinox	0.791	**0.806**	CvsLogEntropy,CvsEntropy CvsLinEntropy	CvsLogEntropy,cbo CvsEntropy
lucene	0.828	0.841	CvsLinEntropy,log-churn-wmc log-ent-numberOfLinesOfCode	CvsLinEntropy,log-churn-wmc CvsExpEntropy
mylyn	0.740	0.740	fanOut,exp-ent-numberOfLinesOfCode lin-churn-numberOfLinesOfCode	fanOut,exp-ent-numberOfLinesOfCode exp-churn-numberOfLinesOfCode
pde	0.748	0.748	CvsLinEntropy,lin-ent-rfc CvsExpEntropy	CvsLinEntropy,lin-ent-rfc CvsExpEntropy

one data set might be useful metrics for another data set.

IG is a metric selection method based on information (Menzies et al., 2007). RF is implemented in R[2], and LTR is implemented in Java. The details can be found in Yang et al.'s work (Yang et al., 2015).

We apply 10-folds cross-validation to evaluate the methods. To conduct cross-validation, we randomly produce the index before splitting and hence all methods use the same training and testing sets each time. The Wilcoxon rank-sum test (Fay and Proschan, 2010) (which is called ranksum for short) at 0.05 significance is used as the statistical test.

5 EXPERIMENTAL RESULTS

The experimental results are reported according to the research questions: Q01, Q02, RQ1, RQ2.

5.1 Q01: Comparison of Best Metrics Selected According to Different Tasks

Since linear regression (Khoshgoftaar and Allen, 1999; Yang et al., 2015) could work well when there are only a few metrics, we use linear regression as the construction method in this subsection. 10 runs of 10-fold cross-validation are performed to achieve results.

[2]http://www.r-project.org./

Table 3 gives the three metrics selected by IG according to different (ranking and classification) tasks and mean FPA of models based on the corresponding metrics. To be noted, the three metrics are not listed in order. The column of 'classification results' gives mean FPA of models based on metrics selected according to the classification task, and the column of 'ranking results' is the corresponding results according to the ranking task. We apply ranksum at 0.05 significance to test whether or not FPA of models based on three metrics selected according to different tasks are significantly different. If they are different, the better results are in boldface.

From Table 3, we can see that even using the same metric selection method, the best metrics selected according to different tasks can be different, which means that the most effective metrics for SDP for the classification task may not be most effective for SDP for the ranking task. In addition, the three metrics selected according to the ranking task achieve models with significantly better FPA over some datasets and no significantly worse FPA over all datasets than those according to the classification task. Therefore, we need to analyze metrics using methods specifically for SDP for the ranking task.

5.2 Q02: Comparison of OMS and IG

The FPA results based on different numbers of metrics selected by IG using the learning-to-rank (LTR) as model construction method have been reported (Yang et al., 2015). In order to compare OMS with

IG, we use OMS to replace IG to obtain results, which are shown in Table 4. The ranksum at 0.05 significance is used to test whether or not models based on partial metrics selected by OMS perform significantly worse than models based on all metrics, but there exist no differences. That is, models based on two metrics selected by OMS work comparably with models based on all metrics over all data sets, while in (Yang et al., 2015), models based on only two metrics work worse than models based on all metrics over Files3.0.

Table 4: Mean FPA Based on Different Number of Metrics Selected by OMS using LTR Approach.

dataset	2	3	5	8	13	21	34	55	89	144	all
Files2.0	0.81	0.81	0.81	0.81	0.81	0.82	0.82	0.82	0.82	0.82	0.82
Files2.1	0.77	0.77	0.77	0.78	0.78	0.78	0.78	0.78	0.78	0.78	0.78
Files3.0	0.79	0.79	0.79	0.79	0.79	0.80	0.80	0.80	0.79	0.79	0.80
Package2.0	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.79	0.77	0.78	0.79
Package2.1	0.77	0.77	0.77	0.77	0.76	0.76	0.78	0.77	0.77	0.77	0.77
Package3.0	0.81	0.81	0.81	0.81	0.80	0.80	0.80	0.80	0.81	0.81	0.78
eclipse	0.83	0.83	0.79	0.80	0.80	0.83	0.82	0.81	0.82	0.83	0.82
equinox	0.79	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80
lucene	0.85	0.85	0.83	0.80	0.79	0.78	0.79	0.79	0.78	0.80	0.80
mylyn	0.74	0.77	0.78	0.78	0.78	0.77	0.76	0.78	0.79	0.79	0.79
pde	0.79	0.79	0.79	0.79	0.79	0.76	0.76	0.77	0.77	0.78	0.76

Since experimental results show that using three metrics selected by both IG and OMS can achieve results as good as using all metrics, we compare models based on only one, two and three metrics selected by them directly in Table 5. The worse results according to statistical tests are underlined.

With one metric, OMS achieves significantly better results than IG over three data sets and no significantly worse results over other datasets. With two metrics, OMS achieves one significantly better result and no significantly worse results. With three metrics, OMS achieves one significantly better result and one significantly worse result than IG. As a whole, OMS performs better than or comparable to IG for most cases. This implies that OMS might be able to better reflect the effectiveness of software metrics for sorting modules in order of defect count than information entropy. Considering that applying IG to ranking problems might need an appropriate split strategy, OMS is a good choice to select metrics for sorting modules in order of defect count.

5.3 RQ1: Appropriate Metric Number and Effective Metrics

A. Appropriate Metric Number

In this subsection, we further investigate the appropriate number of metrics that can achieve competitive results with all metrics for SDP for the ranking task over eleven data sets.

Table 5: Means FPA Results with One to Three Metrics Selected by OMS and IG.

Data	one metric OMS	one metric IG	two metrics OMS	two metrics IG	three metrics OMS	three metrics IG
Files2.0	0.805	0.761	0.806	0.804	0.806	0.805
Files2.1	0.766	0.766	0.766	0.766	0.766	0.766
Files3.0	0.789	0.743	0.789	0.777	0.789	0.789
Package2.0	0.769	0.728	0.769	0.757	0.769	0.763
Package2.1	0.765	0.736	0.766	0.751	0.766	0.777
Package3.0	0.808	0.765	0.808	0.800	0.807	0.815
eclipse	0.829	0.817	0.828	0.822	0.828	0.819
equinox	0.794	0.794	0.793	0.795	0.800	0.808
lucene	0.843	0.813	0.849	0.805	0.848	0.837
mylyn	0.727	0.727	0.737	0.744	0.772	0.739
pde	0.777	0.776	0.785	0.783	0.785	0.782

From Table 4, when using LTR as model construction method, models based on only two metrics selected by OMS work well over all data sets. How about one? In order to answer it, we also compare models based on only one metric (which can be seen from Table 5) with models based on all metrics (which can be seen from Table 4). Unexpectedly, no results are significantly different over all data sets. Over some data sets such as lucene and pde, using one metric even achieves larger mean FPA than using all metrics.

Investigating relationship between metric number and model performance involves two issues: metric selection methods and model construction methods. The above results are achieved using LTR as the model construction method. In order to investigate the effect of model construction methods, we also use RF to construct models, which was demonstrated to perform best over the original data sets (Weyuker et al., 2010; Yang et al., 2015). The results are shown in Table 6. The ranksum at 0.05 significance is used to test whether or not models based on partial metrics selected by OMS are significantly different from models based on all metrics, and the results are underlined if they are significantly worse. Since RF is not suitable for constructing models using only one metric, the results of one metric in Table 6 are actually the results in Table 5, but they are compared to results of RF with all metrics in Table 6.

From Table 6, using RF as the model construction method, models based on only one metric still perform comparably with models based on all metrics over seven out of eleven data sets. This is different from previous observation (Zimmermann et al., 2007) and the case for classification task (Menzies et al., 2007). The results indicates the possibility to find one key metric for sorting modules in order of defect count. However, the results also reflect that most existing metrics are redundant and even noisy.

Table 6: Mean FPA Based on Different Number of Metrics Selected by OMS using RF.

datasets	1	2	3	5	8	13	21	34	55	89	144	all
Files2.0	0.80	0.79	0.81	0.82	0.85	0.85	0.86	0.86	0.86	0.86	0.85	0.85
Files2.1	0.77	0.73	0.76	0.76	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.79
Files3.0	0.79	0.76	0.78	0.80	0.80	0.81	0.81	0.81	0.81	0.81	0.80	0.80
Package2.0	0.77	0.74	0.74	0.76	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77
Package2.1	0.77	0.75	0.73	0.75	0.76	0.76	0.76	0.76	0.76	0.76	0.77	0.77
Package3.0	0.81	0.77	0.78	0.80	0.80	0.80	0.81	0.81	0.81	0.81	0.81	0.81
eclipse	0.83	0.82	0.83	0.84	0.86	0.86	0.86	0.87	0.86	0.87	0.86	0.86
equinox	0.79	0.79	0.79	0.80	0.81	0.82	0.83	0.82	0.82	0.82	0.82	0.82
lucene	0.84	0.81	0.82	0.84	0.85	0.84	0.85	0.85	0.85	0.85	0.86	0.86
Mylyn	0.73	0.73	0.77	0.79	0.79	0.81	0.81	0.82	0.81	0.81	0.81	0.81
pde	0.78	0.76	0.76	0.78	0.78	0.80	0.80	0.80	0.81	0.81	0.81	0.81

In this case, metric selection can not only solve the problem of high dimension, but also improve model performance by deleting noisy metrics.

B. Effective Metrics

In order to investigate which metrics are most effective for sorting software modules in order of defect count, we use Algorithm 1 to rank software metrics over all data sets (the metric with largest FPA values ranks 1). According to original software metrics, these data sets can be categorized into three sets: file-level (File2.0, File2.1,File3.0), package-level(Package2.0, Package2.1, Pack-age3.0) and others (other five data sets). We average the ranks of each category, so metrics with smallest ranks are most effective for the corresponding category. The top ten metrics are shown in Table 7. Eclipse_II include two kinds of metrics: complexity metrics and structure metrics (metrics of structure of abstract syntax trees). The structure metrics are underlined in Tables 7.

From Table 7, total lines of code is useful (ranks 5 and 3 respectively for file-level and package-level data sets). Nevertheless, structure metrics seem to be more effective than the size of modules because the top ten metrics include more structure metrics than complexity metrics, and 'SUM', which is the sum of all structure metric values, is the top one metric for both file-level and package-level.

There exist some metrics with only one value for Eclipse_II (42 structure metrics for package-level and 44 structure metrics for file-level). For file-level Eclipse_II, the metrics 'CompilationUnit' and 'PackageDeclaration' have only one value. However, the corresponding 'Norm' metrics ('NORM CompilationUnit' and 'NORM PackageDeclaration'), have more than one values and have the same ranks as 'SUM'. This is because the values of 'Norm' metrics are values of 'SUM' divided by values for the corresponding original metrics. Their rankings and the rankings by 'SUM' are exactly opposite so their effectiveness for building SDP models for the ranking task is the same. If we use Algorithm 1 to select metrics, we might select metrics that are totally correlated with each other. Therefore, we need to use OMS which considers the redundancy of selected metrics.

For other five datasets (eclipse, equinox, lucene, mylyn, pde), there are six kinds of metrics: change metrics, bug metrics, source code metrics, entropy of changes, churn of source code metrics and entropy of source code metrics. From D'Ambros et al.'s work (D'Ambros et al., 2011), entropy of source code metrics are most effective for constructing SDP models for the ranking task, and then comes churn of source code metrics and change metrics. From our results, all top ten metrics are churn and entropy of source code metrics, so they seem to be consistent with D'Ambros et al.'s experimental results. Nevertheless, by checking metrics that rank behind with mean ranks larger than 190, which are (from the last one with largest mean ranks) churn-noc(Number Of Children), weighted-churn-noc, weighted-ent-noc, lin-churn-noc, exp-churn-noc, log-churn-noc, log-ent-noc, log-ent-dit(Depth of Inheritance Tree),lin-ent-dit, exp-ent-dit, ent-noc, ent-dit,weighted-ent-dit, lin-ent-noc, exp-ent-noc, exp-churn-dit, log-churn-dit and lin-churn-dit, we find that the metrics with larger mean ranks are also entropy and churn of source code metrics. Hence, only part of churn and entropy of source code metrics are effective for sorting modules in order of defect count, which further demonstrates that we should not simply conclude which kind of metrics are more effective.

All top metrics for other five datasets in Table 7 are churn and entropy of linesOfCode(loc) and ResponseForClass(rfc). All metrics that rank behind, which are mentioned above, are churn and entropy of noc and dit. Therefore, the effectiveness of process metrics might be decided more by churn and entropy of which source code metrics than by which churn and entropy of source code metrics. Although source code metrics have been widely debated (Fenton and Neil, 1999), metrics based on changes of source code have been always recommended (D'Ambros et al., 2011; Graves et al., 2000; Moser et al., 2008). From our analysis results, choosing the right basic source code metrics is very important.

To sum up, the experimental results lead to the following findings.

* Models based on only one metric can perform comparably with models based on all metrics over most datasets, which is different from previous observation and the case for classification task.
* The most effective metrics and the most useless metrics can belong to the same kind of metrics (entropy and churn of source code metrics), and we

Table 7: Top Ten Metrics and Their Mean Ranks According to FPA Scores Over Three Sets of Metrics.

File Level	Rank	Package Level	Rank	Other Level	Rank
SUM	1.7	SUM	2.7	log-churn-linesofcode	15.4
NORM-CompilationUnit	2.7	SimpleName	2.3	lin-churn-linesofcode	16.4
NORM-PackageDeclaration	2.7	Totallinesofcode	4.0	log-ent-NlinesofcodeLOC	16.6
SimpleName	3.3	QualifiedName	4.0	lin-ent-linesofcode	18.0
Totallinesofcode	4.7	Block	7.3	log-churn-ResponseForClass	19.2
NORM-TypeDeclaration	5.3	VariableDeclarationFragment	7.7	log-ent-ResponseForClass	19.2
Block	8.7	SimpleType	8.0	lin-churn-rfc	22.0
MethodInvocation	9.0	NullLiteral	9.3	lin-ent-ResponseForClass	22.0
Methodlinesofcode-sum	10.7	MethodInvocation	11.0	exp-churn-ResponseForClass	24.0
Numberofmethodcalls-sum	11.0	SingleVariableDeclaration	11.0	exp-ent-linesofcode	24.6

should not simply conclude which kind of metrics are more effective.

* The effectiveness of process metrics might be decided more by churn and entropy of which source code metrics than by which kind of churn and entropy of source code metrics.
* There exist many redundant and useless metrics among these datasets and we should apply metric selection methods to delete them.

5.4 RQ2: The Same Metrics for Different Data Sets

In the above subsection, experimental results reveal that models based on only one metric can perform well over most data sets. Therefore, in this subsection, we investigate whether one most effective metric can capture all necessary information for sorting modules over all data sets with the same original metrics. According to original metrics, eleven data sets are divided into three categories, as shown in Table 7, and the metric that ranks best in Table 7 is the most effective metric for the corresponding category of data sets, i.e., 'SUM' for file-level and package-level data sets and 'log-churn-loc' for other five data sets. To be noted, the most effective metric for each specific data set, which is shown in the last column of Table 8, is not necessarily the most effective metric for the corresponding category of data sets. We compare models based on the most effective metric for the specific data set and models based on the most effective metric for the corresponding category of data sets (metric that ranks best in Table 7). Ranksum at 0.05 significance is used to test whether they are different. If they are different, the better results are in boldface. The results are shown in Table 8.

In Table 8, models based on the most effective metric for specific data sets perform better than models based on the metric that ranks best in Table 7 over five data sets, which means that there is no single metric that performs well over all five data sets. The most effective metric over one data set might be less effective for another data set. Table 7 can only show that churn and entropy of loc and rfc rank relatively in the front over these five data sets, but the most effective metric depends on the specific data set.

Table 8: Comparison of the Most Effective Metric for Specific Datasets and the Most Effective Metric for All Datasets.

dataset name	using log-churn -loc	using metric selected by OMS	metric name selected by OMS
Files2.0	0.81	0.81	SUM
Files2.1	0.77	0.77	SimpleName
Files3.0	0.79	0.79	Totallinesofcode
Package2.0	0.77	0.77	SUM
Package2.1	0.77	0.77	SUM
Package3.0	0.81	0.81	SUM
eclipse	0.82	**0.83**	CvsWEntropy
equinox	0.76	**0.79**	CvsLogEntropy
lucene	0.80	**0.84**	numberOfBugsFoundUntil
mylyn	0.69	**0.73**	fanOut
pde	0.71	**0.78**	numberOfNonTrivial-BugsFoundUntil

However, using 'SUM' can achieve comparable models over all Eclipse_II data. Although SCC between 'SUM' and the number of defects are not large (less than 0.5 over most data sets), the FPA of models constructed based on only 'SUM' over Eclipse_II are larger than 0.76. It seems that we can use the same metric 'SUM' for these six data sets. Nevertheless, 'SUM' is sum of all structure metric values, and its effectiveness is based on all structure metrics instead of a single metric. Although it is possible to find one metric ('SUM') to effectively sort modules in order of defect count over Eclipse_II data, other metrics are still useful and should be kept. In addition, the effectiveness of 'SUM' implies the possibility of improving metric quality by combining metrics.

To sum up, the experimental results lead to the following findings.

* Using only 'SUM' can lead to good models over all Eclipse_II data. However, 'SUM' is sum of

all structure metric values, and its effectiveness is based on all structure metrics instead of a single metric.

* There is no single metric that performs well over other five data sets. The most effective metric for one data set might be less effective for another one.
* The effectiveness of 'SUM' implies the possibility of improving metric quality by combining metrics.

5.5 Other Discussions

Zimmermann et al. (Zimmermann et al., 2007) computed SCC between complexity metrics and defects over release 3.0 and they pointed out that finding a single indicator for the number of defects was unlikely because the coefficients were not large. However, our experimental results show that only one metric can work well over most data sets although SCC are not large. The main reason is that SCC could not reflect that the ranking for modules with more defects is more important than the ranking for modules with less defects, which has been shown in Table 1. There is no absolute relationship between SCC and the effectiveness of metrics for sorting modules in order of defect count.

In this paper, we adopt FPA as the performance measure, which has been found to equals to $1/(2k)$ plus the area under cumulative lift chart (CLC), where k is the number of modules (Yang et al., 2015). Since k is larger than 300 over all data sets, the values given by them should be approximate. However, the FPA results are very different from the results given by D'Ambros et al. (D'Ambros et al., 2011). To be noted, they actually used $1-(S_{opt}-S_{model})$ instead of area under CLC. S_{opt} is the area under CLC of optimal models, and S_{model} is the area under CLC of actual models. The best results given by D'Amrbos et al. are 0.89, 0.91, 0.86, 0.83 and 0.81 for eclipse, equinox, lucene, mylyn and pde. In order to compare our results with theirs, we transform part of our results: results based on only one metric are respectively 0.89, 0.90, 0.87, 0.78 and 0.83, and the results based on all metrics constructed by RF are 0.92, 0.93, 0.89, 0.86 and 0.86 for eclipse, equinox, lucene, mylyn and pde.

6 THREATS TO VALIDITY

6.1 Threats to Internal Validity

A threat to internal validity of the results presented in this study is the choice of performance measure. There are reasons to choose FPA. For example, FPA was demonstrated to be similar to the area under Alberg diagram (Yang et al., 2015), which has been demonstrated to be reasonable for evaluating SDP models for the ranking task (Ohlsson and Alberg, 1996), and FPA is a performance measure specifically proposed for sorting modules in order of defect count, which has been shown to overcome the shortcoming of 'percentages of defects found in top m modules' (Weyuker et al., 2010). However, when other goals of sorting modules in order of defect count are considered (for example, only percentages of defects found in the top m modules are considered), the choice should be changed, which may lead to a different conclusion.

Another threat is the choice of experimental methods. Cross-validation is applied to obtain results, which are popularly used in SDP domain(D'Ambros et al., 2011), in order to reduce bias. However, there still exists the instability that may affect the conclusions, which can be shown in three aspects. First of all, although LTR and RF can perform better than methods such as regression tree (Yang et al., 2015), LTR and RF are not stable. That is, given the same training data, LTR and RF might obtain different results. Secondly, the parameters for RF are set as suggested by Weyuker et al. (Weyuker et al., 2010) and the parameters for LTR are set as Yang et al.'s work (Yang et al., 2015). For different data sets or different number of metrics, the best parameters might be different, which might lead to different results. Finally, although cross-validation can reduce bias (Khoshgoftaar and Allen, 2003), it still depends on the splitting. When the splitting is different, the results can be different.

The learning algorithms used would be also an internal threat. LTR and RF are used as model construction methods, which have been demonstrated to be best for sorting modules in order of defect count (Yang et al., 2015), in order to obtain reasonable results. However, if a different algorithm is applied, the conclusion might be different.

6.2 Threats to External Validity

The main threat to external validity is data sets. There are not many existing publicly available data sets for sorting modules in order of defect count. Our ex-

periments are based on eleven data sets, including not only the commonly used Eclipse_II (Zimmermann et al., 2007), but also the data sets which include many process metrics (D'Ambros et al., 2010). Therefore, the conclusions are convincing. Having said that, these two sets of data are only a very small part of all data sets, among which there are many data sets based on industrial software systems (Gao and Khoshgoftaar, 2007) that are not publicly available. Other data sets might include different metrics and totally different modules. Therefore, the conclusions over these two sets of data sets might not hold for other data sets.

7 CONCLUSIONS

With more and more metrics introduced, numerous metric selection methods have been applied to investigate the effectiveness of software metrics for classifying software modules into defect-free and defect-prone, while few metric selection methods have been proposed to investigate software metrics for sorting software modules in order of defect count. In this paper, an object-based metric selection (OMS) approach is proposed for sorting module in order of defect count. Experimental results over eleven publicly available data sets show that OMS works better than IG in most cases. Comprehensive empirical studies are also conducted to investigating the effectiveness of metrics over these data sets, and the following findings are obtained, which can help to select effective metrics among existing metrics and direct the introduction of new metrics (for example, churn and entropy of other source code metrics).

1. Different from previous observation (Zimmermann et al., 2007) and the case for classification task (Menzies et al., 2007) , models based on only one metric can perform well over most data sets for sorting modules in order of defect count. However, there is no single metric that performs well over all data sets. Although 'SUM' is good to predict the number of defects over Eclipse_II data sets, 'SUM' is sum of all structure metric values, and its effectiveness is based on all structure metrics instead of a single metric. This implies that no single metric can actually capture all the necessary information for ranking. The most effective metric is data set dependent.

2. The most effective metrics and the most useless metrics can belong to the same kind of metrics, and we should not simply conclude which kind of metrics are more effective.

3. The effectiveness of process metrics might be decided more by churn and entropy of which source code metrics than by which kind of churn and entropy of source code metrics.

4. There exist many redundant and useless metrics (e.g. metrics having only one value) among these data sets and metric selection methods should be applied to delete them.

5. The effectiveness of 'SUM' over Eclipse_II implies the possibility of improving metric quality by combining metrics.

To some extent, the new findings show the status quo of software metrics and help to introduce new metrics for sorting software modules in order of defect count. In our future work, we will focus on introducing new effective metrics for sorting modules in order of defect count.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (Grants No. 61602534) and Fundamental Research Funds for the Central Universities (No.17lgpy122).

REFERENCES

Akiyama, F. (1971). An example of software system debugging. In *IFIP Congress*, pages 353–359.

Basili, V. R., Briand, L. C., and Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761.

Catal, C. and Diri, B. (2009). A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354.

Chidamber, S. and Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

Churcher, N. and Shepperd, M. (1995). Comments on "a metrics suite for object oriented design". *IEEE Transactions on Software Engineering*, 21(3):263–265.

D'Ambros, M., Lanza, M., and Robbes, R. (2010). An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41. IEEE.

D'Ambros, M., Lanza, M., and Robbes, R. (2011). Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, pages 1–47.

Fay, M. and Proschan, M. (2010). Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4:1–39.

Fenton, N. E. and Neil, M. (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5):675–689.

Gao, K. and Khoshgoftaar, T. (2007). A comprehensive empirical study of count models for software defect prediction. *IEEE Transactions on Reliability*, 56(2):223–236.

Graves, T., Karr, A., Marron, J., and Siy, H. (2000). Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering*, 26(7):653–661.

Gray, D., Bowes, D., Davey, N., Sun, Y., and Christianson, B. (2011). The misuse of the nasa metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation and Assessment in Software Engineering, 2011*, pages 96–103. IET.

Hall, M. and Holmes, G. (2003). Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1437–1447.

Halstead, M. H. (1997). Elements of software science. *Amsterdam: Elsevier North-Holland*.

Jabangwe, R., Borstler, J., Smite, D., and Wohlin, C. (2015). Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. *Empirical Software Engineering*, 20(3):640–693.

Khoshgoftaar, T., GAO, K., and NAPOLITANO, A. (2012). An empirical study of feature ranking techniques for software quality prediction. *International Journal of Software Engineering and Knowledge Engineering*, 22(02):161–183.

Khoshgoftaar, T., Gao, K., Napolitano, A., and Wald, R. (2014). A comparative study of iterative and non-iterative feature selection techniques for software defect prediction. *Information Systems Frontiers*, 16(5):801–822.

Khoshgoftaar, T. M. and Allen, E. B. (1999). A comparative study of ordering and classification of fault-prone software modules. *Empirical Software Engineering*, 4(2):159–186.

Khoshgoftaar, T. M. and Allen, E. B. (2003). Ordering fault-prone software modules. *Software Quality Journal*, 11(1):19–37.

Liu, S., Chen, X., Liu, W., Chen, J., Gu, Q., and Chen, D. (2014). Fecar: A feature selection framework for software defect prediction. In *Proceedings of the 38th Anual International Computers, Software and Applications Conference*, pages 426–435. IEEE.

McCabe, T. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Menzies, T., Greenwald, J., and Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13.

Moser, R., Pedrycz, W., and Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *ACM/IEEE 30th International Conference on Software Engineering, 2008.*, pages 181–190. IEEE.

Nam, J. and Kim, S. (2015). Clami: defect prediction on unlabeled datasets. In *30th IEEE/ACM International Conference on Automated Software Engineering, 2015*, pages 452–463. IEEE/ACM.

Ohlsson, N. and Alberg, H. (1996). Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894.

Ostrand, T., Weyuker, E., and Bell, R. (2005). Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355.

Rahman, F. and Devanbu, P. (2013). How, and why, process metrics are better. In *35th International Conference on Software Engineering, 2013*, pages 432–441. IEEE.

Shepperd, M., Song, Q., Sun, Z., and Mair, C. (2013). Data quality: some comments on the nasa software defect datasets. *IEEE Transactions on Software Engineering*, 39(9):1208–1215.

Shivaji, S., Jr., E., Akella, R., and Kim, S. (2013). Reducing features to improve code change-based bug prediction. *IEEE Transactions on Software Engineering*, 39(4):552–569.

Wang, H., Khoshgoftaar, T., and Seliya, N. (2011a). How many software metrics should be selected for defect prediction. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference*, pages 69–74.

Wang, H., Taghi, M., Van Hulse, J., and Gao, K. (2011b). Metric selection for software defect prediction. *International Journal of Software Engineering and Knowledge Engineering*, 21(02):237–257.

Weyuker, E., Ostrand, T., and Bell, R. (2010). Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15(3):277–295.

Yang, X., Tang, K., and Yao, X. (2015). A learning-to-rank approach to software defect prediction. *IEEE Transactions on Reliability*, 64(1):234–246.

Zimmermann, T., Premraj, R., and Zeller, A. (2007). Predicting defects for eclipse. In *International Workshop on Predictor Models in Software Engineering,PROMISE'07*, pages 9–15. Ieee.