A Preliminary Analysis of Publicly Available FLOSS Measurements: Towards Discovering Maintainability Trends *

Ioannis Samoladas
Department of Informatics
Aristotle University of Thessaloniki
542 24, Thessaloniki, Greece
ioansam@csd.auth.gr

Stamatia Bibi
Department of Informatics
Aristotle University of Thessaloniki
542 24, Thessaloniki, Greece
sbibi@csd.auth.gr

Ioannis Stamelos
Department of Informatics
Aristotle University of Thessaloniki
542 24, Thessaloniki, Greece
stamelos@csd.auth.gr

Sulayman Sowe
Department of Informatics
Aristotle University of Thessaloniki
542 24, Thessaloniki, Greece
sksowe@csd.auth.gr

Ignatios Deligiannis
Information Technology Department
Technological Education Institute
54700, Thessaloniki, Greece
ignatios@it.teithe.gr

ABSTRACT

The spread of free/libre/open source software (FLOSS) and the openness of its development model offer researchers a valuable source of information regarding software data. The creation of large portals, which host a vast amount of FLOSS projects make it easy to create large datasets with valuable information regarding the FLOSS development process. In addition initiatives such as FLOSSMole provide researchers with a single point and continuing access to those data. Up to now the majority of datasets from FLOSSMole offered data regarding the development process and not the code itself. From February 2007 FLOSSMole offers data donated by SourceKibitzer, which contain source code metrics for FLOSS projects written in Java. In this paper we provide a premilinary analysis on those data using machine learning techniques, such as classification rules and decision trees. Using the first available data from February 2007, we tried to build rules that can be used in order to estimate the future values of metrics offered for March. Here we present some preliminary results that are encouraging and deserve to be further analyzed in future releases of SourceKibitzer datasets.

*This work is funded by the Greek Ministry of Education (25%) and European Union (75%) under the EPEAK II program Archimedes and partly by the European Community's Sixth Framework Programme under the contracts of projects FLOSSMETRICS (FP6-IST-5-033547) and SQO-OSS (FP6-IST-5-033331)

Keywords

open source public available metrics data, machine learning, classification, decision trees, estimation

1. INTRODUCTION

Research in software engineering has always been in need for data. Performing research inside large software companies, working with a limited set of projects available was the usual way of obtaining those data. Researchers usually couldn't disseminate their data along with their results, since companies were reluctant to allow such publicity. Although some widely available datasets exist (such as the ISBSG dataset) it is rather difficult for software engineering researchers to apply their ideas and methodologies to a large amount of data and different kind of projects. In addition the fact that the majority of the research conducted, involves proprietary software, makes replication rather difficult as datasets are not publicly available. The latter is very important for science, since replication of research allows validation of results.

With the arrival and the spread of Free/Libre/Open Source Software [1] (FLOSS) the software research community has a unique opportunity for obtaining experimentation data. The open development process of FLOSS and the fact that the majority of the projects are hosted under the same provider, has made this data easily accessible. Furthermore there are research groups that extract all the available data from these hosting providers and monitor them regularly. This fact allows research to focus on the analysis of data instead of their collection. Such a group that provides data from various sources is FLOSSMole[2] [2]. The majority of data at FLOSSMole involves the development

[1]With the term FLOSS we are referring to the kind of software that is well known as either "open source software" or "free software" or "libre software" without separating these three terms.

[2]http://ossmole.sourceforge.net

process of FLOSS projects and the description of projects available at the major hosting sites (like Sourceforge.net). Until recently, in FLOSSMole there were no datasets available regarding the source itself, i.e. source code metrics. In February 2007, a site providing metrics results for FLOSS projects written in Java, SourceKibitzer[3] decided to share its data with the research community donating its metrics database to FLOSSMole.

In this study we provide a first analysis on those data utilizing the open source machine learning tool WEKA[6]. We have downloaded the available datasets involving metrics collected in February 2007 and March 2007. We constructed classification models estimating the evolution from February to March for five metrics.

2. DATA DESCRIPTION AND METHODOLOGY

In order to perform our analysis we downloaded from FLOSSMole website the datasets from February and March 2007. These datasets contain the metric results of projects written in Java. From these files only the projects that appear in both of these months were considered in order to participate in the estimation from February to March. The resulted join of the two sets was a sample of 539 projects, appearing both in February and March data sets. From these two files we constructed a third one by subtracting the February values from those of March. These differences were the target of our estimation models.

The metrics available from SourceKibitzer are:

loc Total number of lines including blank lines, executable lines and comments.

cloc Number of lines with comments.

nloc Number of lines that are not comments.

dc Density of comments (i.e. $\frac{CLOC}{LOC}$).

nom Number of methods.

wmc Weighted method count. Sum of the McCabe's Cyclomatic Complexity [3] of all source files.

ncss Number of non commenting source elements (source statements).

npath Sum of NPATH complexity (i.e. number of execution paths) of methods in a package.

fanout Sum of the FANOUT complexity (i.e. the number of classes of a given source file relies on) values of source files.

abstr_coupl Sum of the data abstraction coupling (i.e. the number of instantiations of classes within a source file).

todo_count Sum of the number of TODO comments in a source file. Includes also comments with the words: TO-DO, FIX-ME, FIXME, FIX-IT, FIXIT, XXX, TBD.

[3]http://www.sourcekibitzer.org

bool_exp Sum of the boolean expression complexity of all expressions in a source file (i.e. the sum of the number of logic operators of a given expression).

In order to extract the estimation models we used the Weka machine learning library [6] [1]. The data used for the analysis (independent variables) were the February data, while dependent variables were the differences between February and March values. Some descriptive statistics regarding the February values are shown in Table 1.In total 539 projects were considered. The new variables that represent the differences in values between the two months are denoted with a DD prefix.

3. MODEL AND ANALYSIS DESCRIPTION

The goal of our analysis is to model the fluctuation of variables nloc, nom, ncss, npath and fanout from February to March. These variables are selected because they are considered to affect a critical quality aspect of software, namely its maintainability. In particular decision trees and rules will be used to estimate the difference of the values of the above variables observed during March given February values. The methods used provide estimates in the form of interval values and consequently the values of nloc, nom, ncss, npath and fanout were discretised according to Table 2. The variables were discretized using the SPSS discretization function. Initially we aimed at equal frequency binning which was not possible due to the granularity of data. Finally three intervals were considered automatically by SPSS for each variable the first interval contains negative values, the second interval values close to zero and the third interval conatins relatively high values. Negative values refer to code entities removed (e.g. deleted lines of code). In practice, Interval (Category) 1 represents variable decrease, Interval 2 represents variable stability and Interval 3 represents variable growth. The models are produced by two symbolic algorithms C4.5 (J48) [4] and RIPPER (JRIP)[5]. The first algorithm outputs a decision tree, while the other outputs a set of classification rules.

The decision tree model J48 consists of an hierarchy of univariate binary decisions. The algorithm used operates by choosing the best variable for splitting data into two groups at the root node. It can use any one of several different splitting criteria, all producing the effect of partitioning the data at an internal node into two disjoint subsets in such way that the class labels are as homogeneous as possible. This splitting procedure is then applied recursively to the data in each of the child nodes. Finally, a decision tree is produced and specific branches of this tree are pruned according to the stopping criteria, so as to avoid overfitting of the data and over-specialization of the model.

RIPPER is a rule induction algorithm that provides classification rules. Each rule has a *body*, which consists of one or more conditions under which the rule will fire, and a *head* which consists of the predicted class of faults. These algorithms learn one rule, remove the examples that this rule covers and proceed with the next rule. Any remaining uncovered examples, are handled by a *default* rule that fires without any conditions and predicts the most frequent class among the remaining examples. This also implies that the

Table 1: Descriptive statistics of the estimated variables for February values

Variable	Descriptive Statistics			
	Minimum	Maximum	Mean	Standard Deviation
nloc	29	1296302	53179.249	105400.071
nom	1	86577	3319.803	6406.675
ncss	8	652473	24554.621	49398.615
npath	1	2147483647	280526718.413	661792242.014
fanout	0	4322	2595.493	4718.206

Table 2: Intervals estimated for each variable

Variables		Category		
Name	Original Variable	1	2	3
DDnloc	nloc	[-132287, 0)	[0, 91)	[91, 29497]
DDnom	nom	[-8039, 0)	[0, 4)	[3, 1645]
DDncss	ncss	[-62891, 0)	[0, 35)	[35, 14750]
DDnpath	npath	[-2147431420, 0)	[0, 1)	[1, 983729144]
DDfanout	fanout	[-2676, 0)	[0, 4)	[4, 1046]

rules are presented in the order that they are discovered, and during execution they are considered in this order. The methods are further described in [1], [4].

Both of the above methods are applied using the default parameters as implemented in Weka except for the minimum number of items in each class which is increased to 10. This is done to avoid overspecialization of the model to the data, as the training set has relatively many observations.

The evaluation method used to assess the estimation accuracy of the models is 10 folds cross validation. The 10-fold cross-validation process splits the data into 10-equal disjoint parts and uses nine of these parts for training the classification framework and one for testing. This is done 10 times, each time using a different part of data for testing. The training data are used to train a classification algorithm. The learned model is then applied to the test data. Then according to the classification accuracy of the model we calculate hitrate which is the ratio of the number of metric values estimated to the correct interval to the total number of estimations.

4. RESULTS

For the estimation of the DDnloc the model derived from the whole set of data is presented in Figure 1. The values of abstract coupling of February are used to estimate the fluctuation of the nloc of March values. The tree of Figure 1 can be interpreted as following:

- If the abstr_coupl is equal or less than 1159 then DDnloc is estimated to be in the second interval. This leaf classifies totally 438 projects from which 85 are incorrectly estimated to the second interval.
- Otherwise if abstr_coupl is greater than 1159 DDnloc will be in the third interval. This leaf classifies 100 projects from which 34 are misplaced to the third interval.

As mentioned 10 folds cross validation is used for the evaluation of the model. The model succeeds 75.84% hitrate in this

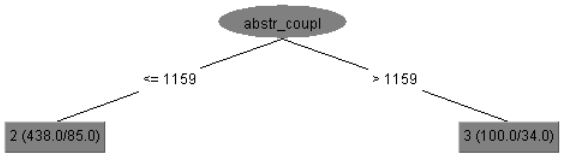

Figure 1: Decision tree for the estimation of DDnloc

case. JRIP algorithm suggests two rules for the estimation of DDnloc, presented in Table 3:

The first rule states that if ncss value is more or equal to 14726 and abstr_coupl is more or equal to 1182 then DDnloc will be in the third interval. This rule classifies 100 projects from which 34 are misclassified. The second rule classifies the remaining438 projects to the second interval being inaccurate in 85 cases. The hitrate of the model is 75.84%.

The next variable participating in this preliminary analysis is DDnom. The tree (Figure 2) suggests abstr_coupl, todo_count, loc and dc as the variables used to classify the increase in the number of methods into an interval. This tree presents 76.2% hitrate.

The rules used to classify DDnom are presented in Table 3. Rules point out fanout, todo_count, dc and abstr_coupl as the major attributes that can classify DDnom. The hitrate of the rules evaluated with 10 folds cross validation is 75.84%.

The tree for DDncss variable is presented in Figure 3. The hitrate of the tree model is 76.95%. abstr_coupl, todo_count and dc classify DDncss into an interval. Rules on the other hand use the values of ncss of February to estimate the DDncss for March along with abstr_coupl. The suggested rules has 75.28% evaluation hitrate and is presented in Table 3.

Table 3: Variable estimation rules

Variable	Rules
DDnloc	$(ncss \geq 14726)$ *and* $(abstr\_coupl \geq 1182) \Rightarrow DDnloc = 3(100.0/34.0)$ $\Rightarrow DDnloc = 2(438.0/85.0)$
DDnom	$(fanout \geq 3086) \Rightarrow DDnom = 3(126.0/49.0)$ $(todo\_count \geq 30)$ *and* $(dc \leq 0.1352)$ *and* $(abstr\_coupl \leq 716) \Rightarrow DDnom = 3(33.0/12.0)$ $\Rightarrow DDnom = 2(379.0/53.0)$
DDness	$(ncss \geq 14726)$ *and* $(abstr\_coupl \geq 1182) \Rightarrow DDncss = 3(100.0/33.0)$ $\Rightarrow DDncss = 2(438.0/82.0)$
DDnpath	$(abstr\_coupl \geq 247)$ *and* $(npath \leq 2129242501)$ *and* $(loc \geq 79814) \Rightarrow DDnpath = 3(68.0/22.0)$ $\Rightarrow DDnpath = 2(470.0/117.0)$
DDfanout	$(todo\_count \geq 45)$ *and* $(fanout \geq 3422) \Rightarrow DDfanout = 3(95.0/29.0)$ $\Rightarrow DDfanout = 2(443.0/82.0)$

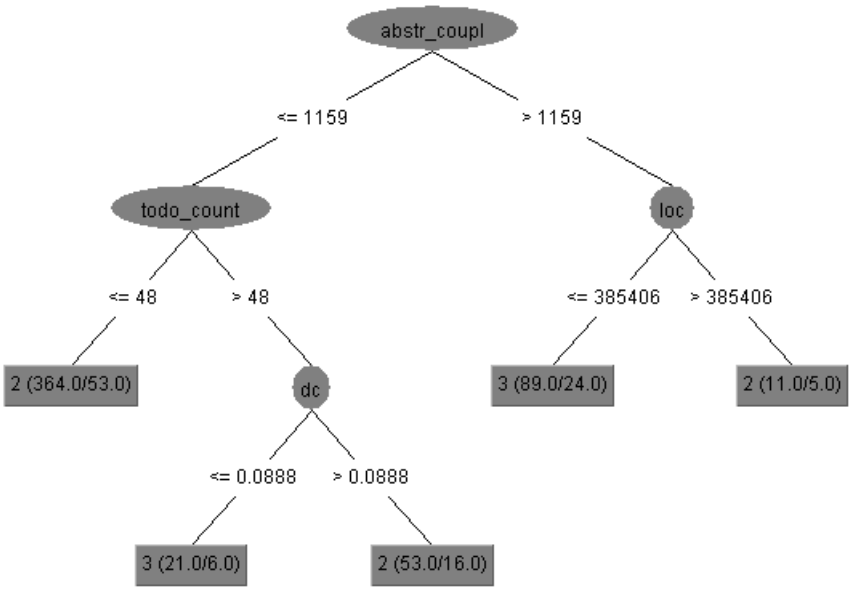

Figure 2: Decision tree for the estimation of DDnom

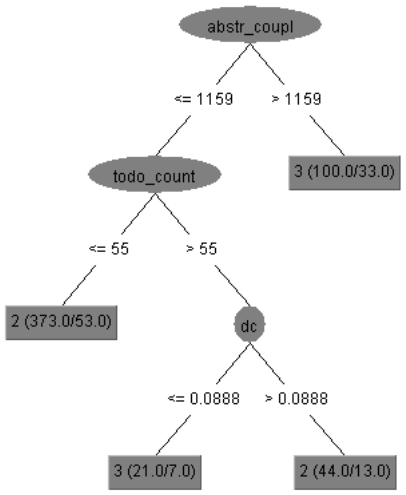

Figure 3: Decision tree for the estimation of DDncss

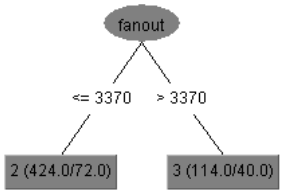

Figure 5: Decision tree for the estimation of DDfanout

The estimation of DDnpath (Figure 4) by J48 algorithm was a difficult task as many variables participated in the estimation of the particular variable such as npath, ncss,loc, dc, todo_count with relatively low accuracy (69.33% hitrate). On the other hand rules consider only abstr_coupl, npath and loc as the most important variables in the classification of DDnpath. The rules of Table 3 present 71.75% hitrate.

The tree for estimating the difference in the values of fanout observed during March uses only fanout values of February. The tree has 77.51% hitrate and is presented in Figure 5. JRIP rules apart from fanout uses the todo_count values to estimate DDfanout. The rules of Table 3 present 78.25% estimation accuracy

5. CONCLUSIONS - FUTURE RESEARCH

In this paper we present a premilinary analysis on publicly available measurements from SourceKibitzer using classification models. The analysis produced trees and rules that estimate the fluctuation of certain metrics. The results were encouraging and the models built estimated future values of metrics with a relatively satisfactory accuracy. This kind of analysis, in this application level, enables us to:

- perform less measurements or measure less frequently
- identify future problematic situations (e.g. we can predict that a FLOSS application that we consider for the first time, will present a high value of fan out or complexity and will probably face maintainability problems in future releases)

Apart from the application level metrics, SourceKibitzer offers detailed file level metrics. Our intention is to apply similar models at the file level, to allow the identification of fault prone files. Future research also includes verification of the model with new metrics releases from SourceKibitzer and finding and validating relationships of individual or composite metrics with external, high level, user specific quality attributes of the FLOSS software examined.

6. REFERENCES

[1] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[2] M. Conklin, J. Howison, and K. Crowston. Collaboration using ossmole: a repository of floss data and analyses. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM Press.

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

[4] R. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, 1993.

[5] I. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 1999.

[6] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.

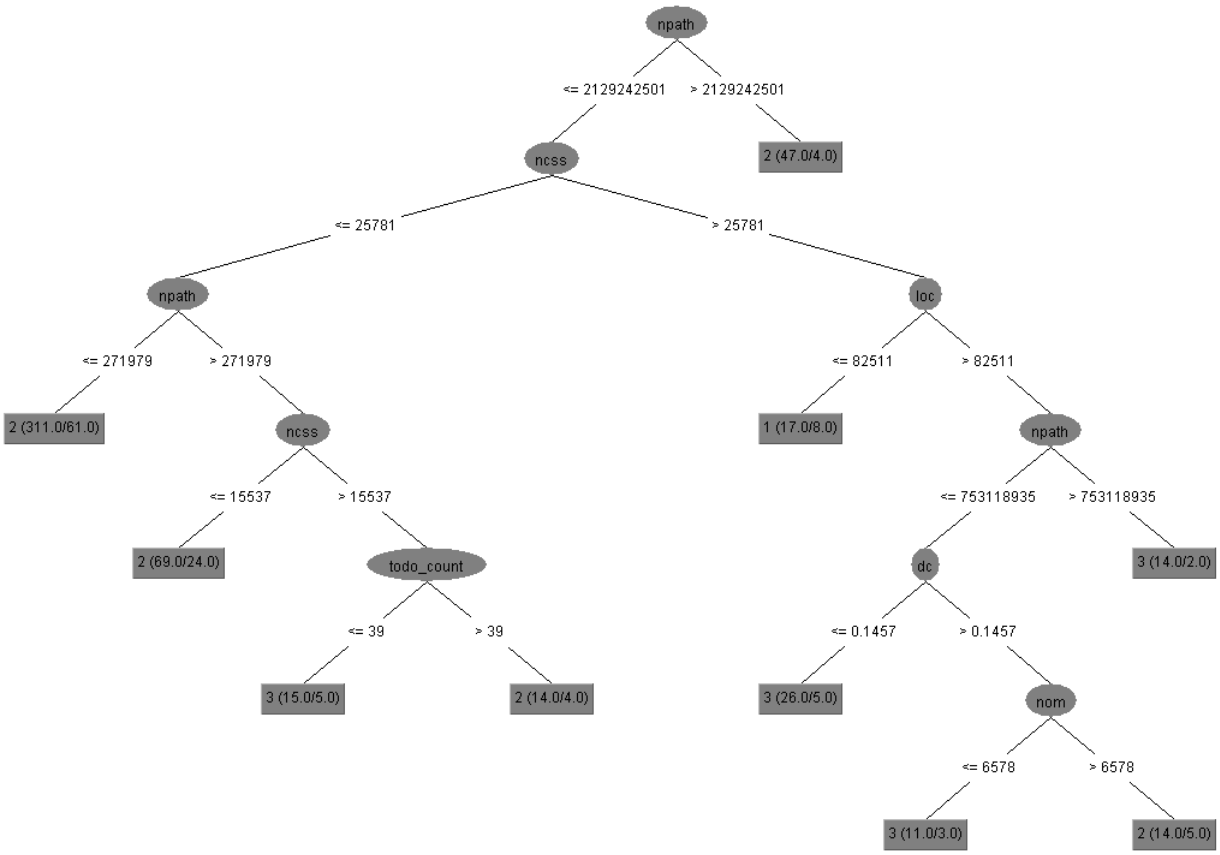

Figure 4: Decision tree for the estimation of DDnpath