Characterizing Developers' Rework on GitHub Open Source Projects

Thiago R.P.M. Rúbio and Carlos A. S. J. Gulo

LIACC / DEI, Faculdade de Engenharia, Universidade do Porto,
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal,
{reis.thiago,prodei1300766}@fe.up.pt

Abstract. Identify Rework on Open Source Software (OSS) projects is a challenging, complex and still open issue. We believe that in OSS environment, team members should introduce less rework than volunteers, due to their knowledge about the project, but how to characterize and classify developers' actions as rework sources? In this paper we presented a novel approach for classifying commits by analysing data from open source repositories, named in GitHub. We constructed our classifier following a data mining methodology and comparing the performance of three algorithms: Decision Trees, Naive Bayes and K-NN. For testing our model, we applied data of 3311 commits from the Subversion project. Results show that K-NN outperforms the others in this task and the model can predict whether a commit is a rework source or not with 70% of accuracy. Our training data depends much on the relation commits-issues and and the collecting methodology should be improved. We envisage to create a more general classification model by using a bigger dataset including data from many other projects.

Keywords: Rework, Data Mining, Classification, Open Source, OSS, GitHub

1 Introduction

Software development is facing a big change. Traditionally, companies have teams that work under a certain budget to achieve milestone objectives. On the other side, the Open Source Software (OSS) has its importance growing in the last years and is changing the rules of this game [1]. The Open Source initiative is based on free code which any other developer can contribute with the project by free will.

One big problem in this scenario is the work introduced by issues not solved consistently or bugs created by developers. This work was not planned in project specification and is called Rework. Rework can be defined as the time and effort considered in trying to fix some bug or solving an issue that was already considered solved. It implies big costs because of the incremental nature of software. Earlier a problem is found, less is the cost to be solved [2].

In OSS software, finding rework sources could help to discover developers' behavior and create a profile for each of them, making it easy to select the best team members or even select those who need help to improve their programming. In fact, it helps to reduce costs, improving the quality of code. Identifying and reducing rework is fundamental for decreasing the cost of development and maintenance of the software [2]. Companies are also interested in this because they could benefit mixing their traditional techniques with ours and reduce even more their expenses with rework. Traditional development has a much more controlled environment and guided by project budgets, they have their own processes for dealing with rework. In OSS software the main difficulty is the lack of knowledge about the quality of code coming from different developers with very different profiles [3]. The impact of rework in OSS software should be studied as a way to know and compare how open source projects are affected by rework.

In this paper we present a data mining approach to create developers' profiles and classify their actions based on the observation of real open source projects development data. We have created a general model, using well-known algorithms and evaluated their performance to obtain a good prediction.

Using a data mining methodology we have compared several algorithms and selected the best model for the classification task, expecting to improve the quality and the reliability of the predictions. We believe that our model can be a good ally on identifying rework sources in projects and helping developers to manage new code. This work had three main goals (i) use data mining tools and methodology to create a working model for classifying real OSS data; (ii) compare the performance of classification using different algorithms and select the best model (iii) analyse the model's predictions and evaluate developers' profiles in OSS projects.

Analysing a real world project's source repository is a big challenge, but also the results are very interesting [4]. Outcomes from our model included the characteristics of the roles members and volunteers and their relation to the creation of rework. Although the type of developer was very important, we wanted to discover more about the relation between rework and other features on developer's actions. Our results proved to be very surprising. We have found that members do also introduce a considerable amount of rework, but volunteers work in simpler tasks.

The rest of this paper is structured as follows: In Section 2 we present the Open Source platforms scenario. Section 3 discusses the problem of based on reviewing the literature introducing rework in software projects and how establishing a developer profile could help reducing efforts in tackling software bugs. We also describe the GitHub environment as an example of OSS environment. In Section 4 we describe the experimental design of our classification model. Section 5 presents the experimental evaluation of the model. We discuss the findings of this work and point lines of future research in Section 6.

2 Related Work

Open Source Software (OSS) is a trend in software development. Since late 1990, an uncountable number of projects have grown in this environment proving it can be successful and and profitable [5]. Research interest in OSS is very diversified. In some way, OSS platforms represent a social networking model between developers and customers, useful information for analysing software marketing and customers preferences [6]. In other way, people enrolled with OSS highlight the relationship between the customers as volunteers and the satisfaction with the product [4]. Big software companies (e.g., Microsoft, Google GNOME, Linux, etc.) are also focusing OSS projects in the last years motivated by the collaboration between volunteers and developers, which can reduce costs, spread the technology and help developing more user-guided tools.

Given its free nature, OSS create a difficulty in managing resources, planning and delivering projects. As said in [7], resource allocation and budgeting is even a harder challenge. The cost of the development is an important factor for the success of a project [7] and here we find the rework cost. As explained in section 1, rework consists on the effort and consequently monetary cost of trying to fix something that was already considered a closed issue. Rework is, in fact, a big problem in software engineering, consuming big part of the project budget (40% up to 70%) [8]. Introducing rework could be explained by human problems in project management like communication, formation and work conditions [8] and the Industry believes that great part of rework could be early identified and avoided, but until now not much attention has been paid in studying rework.

We point out the relation between rework and code quality, measured regarding the actions performed by developers, named commits. Commits are the registries of the work and have the information of all modifications in previous files and new code related to some issue discussed by the team. Commits provide us the information about how work was done and that could be used for predicting if a commit represent or not a rework source.

3 Problem Statement

In a software project, all commits must be indexed by an issue and usually, software development includes two types of tools for managing this: a SCM (Software Configuration Management) tool and an Issue Tracker. SCM tools are responsible for storing all the code produced and all the information about modifications made in the development process (commits). Issue Trackers are responsible for project management, storing the issues and their relation with commits. In Issue Trackers we can see the history of the project and how commits effectively concludes each activity needed.

Our work was to merge the information in this two sources and analyse the data of a real project. We choose the GitHub [1]platform because of its great importance in OSS environment. Known as the largest open source community, GitHub is one of the most important code hosting sites and has a successful social component useful for analysing developers' behavior [9,10]. Based on the Git version control system, GitHub hosts all kinds of open source projects, with

[1] GitHub.com

different sizes and programming languages. Big companies have a very important place in GitHub as they carry large projects and high numbers in the platform.

We have selected the Apache Subversion (SVN) [2] project by its relevance and importance in GitHub platform. A software versioning and revision control system distributed as free software under the Apache License. Recognized as the sole leader in the Standalone SCM category, Subversion has in GitHub about 50000 commits divided into 768 branchs and 238 defined milestones. Fifteen developers are considered members of the actual development team and the average development productivity in Subversion is about 300 commits and 140 files modified by 8000 lines of code per month. Subversion also uses the Tigris[3] platform as the Issue Tracker system and have five types of issues: DEFECT, CLOSED, NEW, REOPENED, and STARTED, in a total of approximate 2690 activities. This system provides us the information regarding which commits are related to these issues and their committers.

4 Model and Experimental Setup

In order to analyse Subversion, we have prepared an experimental setup which include gathering the data in GitHub and Tigris, perform some queries to join and prepare this information, a statistical analysis about the content in hands and selection of the relevant features for our model. Finally we created our best classification model comparing a range of classification algorithms and parameter refinement. We evaluated the performance with each algorithm tested. In our data we found some commits that clearly have indicated rework and others that not. Our work was then to train the model with the first set and test it with the unlabeled data. Our main task was the classification of commits as rework sources or not, according to the information about the developer's activity.

4.1 Data Gathering

The data mining activity started by collecting the data. Two very helpful tools were used in this job: CVSAnaly and BICHO. These tools are part of the MetricsGrimoire project and fundamental for the OSS research in FLOSSMetrics project [4]. The tools operate similarly: given a project URL they make automatically all the process of crawling the repository and collecting available (and public) data. Then a SLQ relational database was created mapping the relation between the content properties in tables.

With the tools we have collected around 100350 commits, comprehended between July, 2007 and December 2014. There were 33 developers enrolled with the project (committing) in this period, 15 labeled as members. The original data description gathered is listed on Table 1.

[2] Apache Subversion - https://subversion.apache.org/
[3] Subversion Tigris - http://subversion.tigris.org/
[4] FLOSSMetrics Consortium. (2012)

Table 1. Data description - Data originally gathered

Variable	**Type**	**Value**	**Sample**
commit_id	id	integer	189
type	atribute	text	M
file_id	atribute	integer	3
file_name	attribute	text	main.py
file_path	attribute	text	branches/remove-log-addressing/subversion
file_parent_id	attribute	text	4
revision	atribute	integer	100978
date	attribute	date	2014-07-01 04:38:02
code	attribute	text	yes
message	attribute	text	Follow-up to r1639319: Fix file object lifetime.,*
committer_id	attribut	integer	16
committer_name	attribute	text	stefan.fuhrmann
member	attribute	text	no
rework	target	text	yes/no/?

The main item here is a commit, referenced by the commit-id identifier and composed by the id of the developer, its type (member or not), date, revision etc. The field type comes for the type of commit, mapped by CVSAnaly as one of the types: A) ADDITION – file added; D) DELETION – file deleted; M) MODIFIED – file modified and C) COPY - file copied

In this stage we also gathered the information about the issue, namely searching from the commit-id in the issues table and collecting the type of activity. We created a rework column that is used as the label column. When the type of the issue is DEFECT or REOPENED we automatically find the correspondent commit-id responsible for the error and set the rework as true. In the other cases it is set false.

We ran a few statistical analysis on RapidMiner to evaluate the information collected. Figure 1 offers the distribution of the commits between all committers, an interesting behavior that shows how developers contribute. Figure 2 in other hand shows the distribution of the commits by type of commit. We clearly conclude that the action "M" is the most common as it represents code submission.

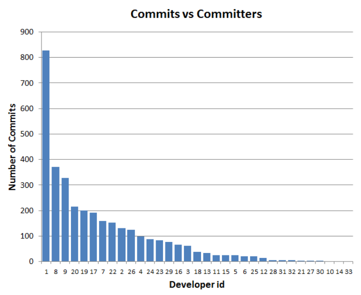


Fig. 1. Distribution of the commits by developers

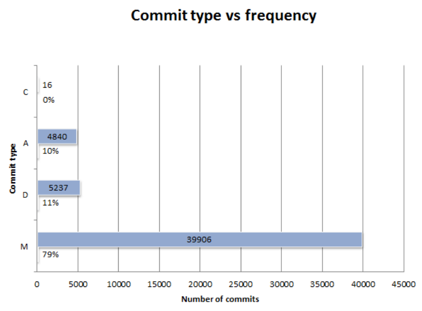


Fig. 2. Commits per type: A) add; D) delete; C) copy; M) modify

We highlight that a great number of commits in the table was not referenced by an issue. In our analysis, only about 35000 of the 100350 commits (near 34%) could be labeled with this approach. This is an important factor for our study, since it means we have a large set of unlabeled data for testing and a correspondent training set (labeled) not so big.

4.2 Data preparation

Using the RapidMiner [5] framework, the next step in the process was conduced. The statistical data analysis implied that not all information contained in the tables was very relevant to our study, and other needed to be prepared. Figure 3 highlights that revision, date, file-path, file-parent-id, committer-name and message seems to have no strong correlation with the rework cases. Although still in Figure 3 we clearly see that the type of the developer was a good property for this. Thinking about the semantics of the rework, and the statistical analysis, the most important features were selected: type of developer, type of modification and number of files affected by one commit. In fact, this information was not present in our initial data, but is a result of counting the files affected by the same commit.

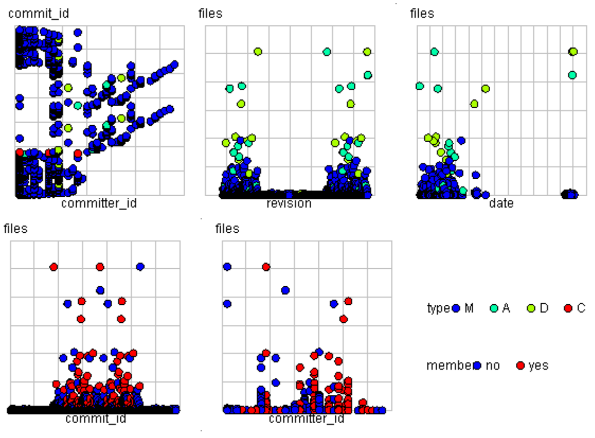


Fig. 3. Statistical visualization of the features

Regarding to data cleansing, we have found some inconsistent commits that were removed. Some had incomplete fields (the tools used do not make all fields mandatory and we encountered many null values that could induce an error in our evaluation), others were identified as **merge** commits (a commit that make

[5] https://rapidminer.com/

the new code available as the official working branch) and were not referenced by activities. There were few outliers identified and we believe that this could be explained by a previous filter in the gathering tools. To solve this situation we have created some basic SQL scripts that filter the undesirable behavior.

Here, our working dataset was consisting in 3311 commits, resulting from the data preparation. We divided it into two subsets: training (containing all labeled samples) consisting of 1490 samples and test (unlabeled) consisting of 1821 samples. We used the training subset for all modelling activities and the test subset for the final evaluation of the model.

4.3 Modeling

In this section, we present the creation of the prediction. More than creating a simple model, we wanted to compare how different algorithms could impact on the results and the knowledge over the data. Using RapidMiner, we've chosen three well-known algorithms for the prediction task: Decision Trees, Naive Bayes and K-NN. These are powerful yet simple mechanisms used in data mining for classification. Moreover, the chosen algorithms needed to be flexible on parameter types and resilient enough for the size and complexity of our dataset. For all three algorithms we have followed the same methodology for improving the model and check performance.

We have started with a simple algorithm model in RapidMiner and applied it to our training set. This implementation gave us a first look at the computed data and a comparative output for improving the performance. The main problem of this approach was that it lacked a training mechanism and could be biased by the distribution of the data. The following approach was improve it by using a 70-30 split mechanism. We sampled the training set using stratified sampling and added a performance validation tool in order to compare results and refine parameters. The final configuration results on different approaches for training, were: Bootstrap, Split-Validation, Cross-Validation and Holdout.

Decision Trees We have first applied the above methodology to the Decision Trees algorithm. We have chosen this first because of the meaning of its results. The algorithm creates a representation of the tree that is the model learned. With Figure 4 we observed the resulting model as the tree output. We could see that our first hypothesis was someway correct as decision weight was heavier on the number of files, membership and type of modification. As we thought, a commit that alters a big number of files have higher probability to introduce rework. Equally, we observed that a commit changing files tends to introduce more rework than a commit removing or adding files, as the last operations were not too frequent as the first.

Following our strategy, we have implemented the other configurations. We have tried different parameter settings and verified that the best setup is accuracy as the parameter criterion and maximal depth 6. Table 2 show the performance evaluation for each configuration. We observed that the results were not

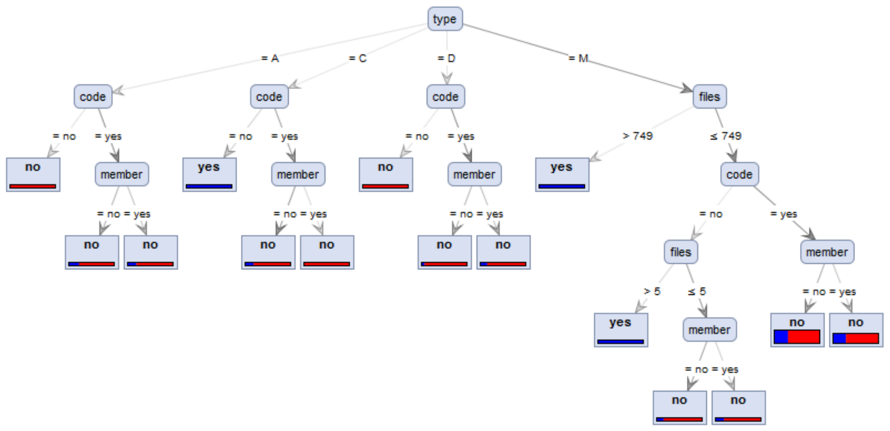


Fig. 4. Decision Tree resulting model

so divergent, which could be a sign that our data represents well the big picture of the project repositories or it that this data do not allow great improvement.

Naive Bayes Differently from Decision Trees, Naive Bayes is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features. As made with Decision Trees, we have create a simple model and then tried different configurations with Naive Bayes. Table 2 shows the performances of the this algorithm, when applied to our training set and compared to the previous.

K-NN Finally, we have tested the K-NN algorithm. The k-Nearest Nearest Neighbors algorithm is a non-parametric method used for classification and regression that consists in searching the belongings of a sample by consulting the weights of it K-nearest neighbors. We used parameter k = 2 with weighted vote and Mixed Euclidean Distance. Back in Table 2 we could observe that this algorithm clearly outperformed the others.

4.4 Performance and Final Model

In data mining, evaluating performance is very important to verify if our predictions are as good as we want to. Performance is the measure of the correct predictions compared to the errors. Our model was based on identifying commits as rework-sources (or not). We have a true positive when we identify a rework-source as such; not identifying it as so causes a false negative.

From the relationship of these values we inferred how good our classification was using several standard metrics. In our work we used the most relevant

Table 2. Performance evaluation for different setups and different algorithms

Strategy	**Accuracy (%)**	**Recall (%)**	**Precision (%)**
Decision Trees			
Simple Model	69.80	0.22	100
Bootstrap	69.73	0.44	100
Split Validation	69.80	0.22	100
Cross Validation	69.80	0.22	100
Holdout	69.87	0.44	100
Naive Bayes			
Simple Model	70.00	0.44	61.11
Bootstrap	70.00	0.68	60.30
Split Validation	70.00	0.55	100
Cross Validation	70.20	1.48	100
Holdout	70.00	1.71	61.49
K-NN			
Simple Model	75.70	5.19	33.89
Bootstrap	78.52	5.48	36.64
Split Validation	75.70	5.19	31.89
Cross Validation	75.70	5.19	37.34
Holdout	73.83	1.41	100

ones: precision, recall, and accuracy. These values depend on the threshold value obtained using a ROC curve. Figure 5 offers the ROC curves for the three algorithms of the setup. The best threshold depends on the algorithm and by our analysis we inferred that for Decision Trees is 0.42, and for the other two algorithms is 0.5.

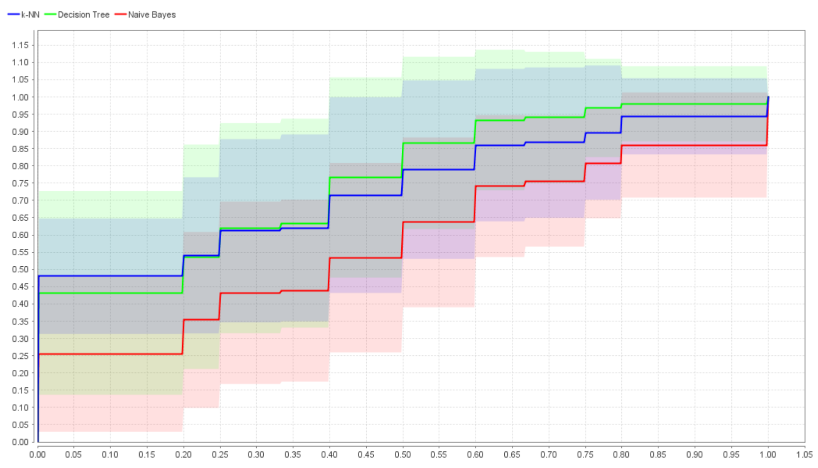


Fig. 5. ROC curve comparison for the threshold

In our problem, accuracy was our main performance metric used to select the best model, our final configuration for the classifier. Accuracy was calculated as seen in Equation 1

$$Accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{1}$$

and provided a measure to determine the quality of the classification criteria. We have finalized this modeling stage by selecting the K-NN classifier with Bootstraping validation as our final model setup.

5 Empirical Evaluation

We have seen how the final classifier model was created and selected, altogether with the comparison of different algorithms using RapidMiner. But as said in section 4.1, the training subset corresponded to 34% of the total number of commits and the other 66% of the data was called test subset. This was the data we wanted to classify. We wanted to apply our model to this data and check the knowledge that extracted from it. Our ultimate goal was to verify if there were significant developer profiles with different behavior concerning the introduction of rework in OSS code.

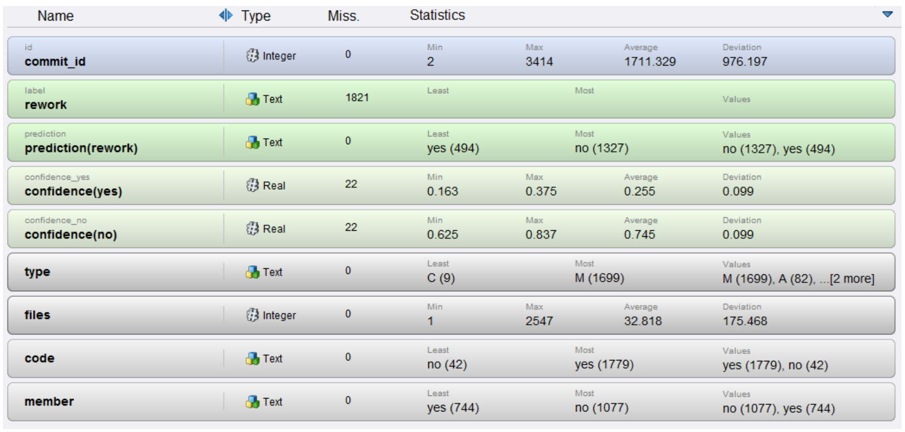
Table 3. Payoff Matrix for the model

	True yes	**True no**	**Class Precision**
Pred. yes	1582	3347	32.10%
Pred. no	1788	4295	70.61%
Class Recall	46.94%	56.20%	

Table 3 shows the training payoff table for the model. The results were obtained applying the classifier to the unlabeled data and condensed in Figure 6. At total, 1821 commits were classified and 494 were predicted to be rework-sources, corresponding to a 27% of the total. Considering that our prediction was realistic, this is a very interesting result. It confirms that rework exists in this environment and seems to achieve the 30%-40% of cost estimated by the industry [8].

Manipulating the data we queried other relevant information. We have discovered that in the total of 494 problematic commits (possible rework sources), 200 (40%) would be introduced by members and 294 (60%) would be in hands of volunteers. From the point of view of the type of rework, 483 (98%) would be in coding activities, facing 11 (2%) in other kinds.

We have found that volunteers introduced more rework in project than members, corroborating the hypothesis that they may not know much about the code specificity. But it is interesting that members of the team introduced a remarkable quantity of rework also. In fact, this is mostly related to the complexity of



Name	Type	Miss.	Statistics			
id **commit_id**	Integer	0	Min 2	Max 3414	Average 1711.329	Deviation 976.197
label **rework**	Text	1821	Least	Most		Values
prediction **prediction(rework)**	Text	0	Least yes (494)	Most no (1327)		Values no (1327), yes (494)
confidence_yes **confidence(yes)**	Real	22	Min 0.163	Max 0.375	Average 0.255	Deviation 0.099
confidence_no **confidence(no)**	Real	22	Min 0.625	Max 0.837	Average 0.745	Deviation 0.099
type	Text	0	Least C (9)	Most M (1699)		Values M (1699), A (82), ...[2 more]
files	Integer	0	Min 1	Max 2547	Average 32.818	Deviation 175.468
code	Text	0	Least no (42)	Most yes (1779)		Values yes (1779), no (42)
member	Text	0	Least yes (744)	Most no (1077)		Values no (1077), yes (744)

Fig. 6. Results of prediction on unlabeled data

the activity, as members usually have commits with high number of file modifications. Volunteers seemed to choose simpler issues, in which they have lower probabilities to introduce rework. 20 of the 33 developers (60%) introduced rework, and the distribution of the problematic commits is normalized.

The results have shown a clearly coexistence of two profiles of developers by analysing the classification of the commits. We attribute this profile distribution into two distinct roles of developers: (i) regular developers which, members or not, have probability of introducing bugs or other rework in the project and (ii) leader developers, expert in programming techniques with a very good knowledge about the code.

6 Conclusions and Future Work

Rework is a common problem in Software Engineering and some questions about it are open opportunities of research. This work presented a novel approach for the rework problem. Based on data mining we have created a model that can classify developers' actions and predict if they will lead to rework. Moreover, our model could give some precious information about developers' behavior and associate profiles.

The model was developed following a data mining methodology and in the process we have illustrated all the stages from data preparation to performance comparison. Three different algorithms were compared, named Decision Trees, Naive Bayes and K-NN. The results show that the Decision Tree algorithm is a very good way to understand the process, but the best algorithm in our case was K-NN. It outperformed the others in terms of accuracy, recall and precision.

Although volunteers introduced more rework than members confirming our expectations, results have shown that members also presented this behavior. We have discovered that volunteers usually chose to contribute with more punctual activities, with low impact on the source code. Regarding the classification of commits, we have observed that the most important features for the classification were the number of files affected, the type of action and the role of the developer. Other features such the management of the rework inside project's versions and the impact of the contribution of volunteers in many projects at the same time are beyond the scope of our model.

We consider this work a first attempt to tackle the problem of calculating rework on OSS and to analyse the characteristics and impact of developers in open development. We expect to expand this study in three main ways 1) creating a more general model that applies to all open source repositories and have a better tunneled parameters; 2) managing to develop new algorithms and combined strategies for multi-label classification; 3) integrating this approach in an online tool that could help developers at the moment of merging code.

7 Acknowledgements

The work of Thiago Rúbio has been funded through a IBRASIL Grant. IBRASIL is a Full Doctorate programme selected under Erasmus Mundus, Action 2 – STRAND 1, Lot 16 and coordinated by University of Lille. The second author thanks Universidade do Estado de Mato Grosso (UNEMAT - Brazil), for the support given.

References

1. Gaff, B.M., Ploussios, G.J.: Open source software. Computer **45**(6) (2012) 9–11
2. Boehm, B.: Software risk management. Springer (1989)
3. Zhao, X., Osterweil, L.J.: An approach to modeling and supporting the rework process in refactoring. In: Software and System Process (ICSSP), 2012 International Conference on, IEEE (2012) 110–119
4. Von Krogh, G., Haefliger, S., Spaeth, S., Wallin, M.W.: Carrots and rainbows: Motivation and social practice in open source software development. Mis Quarterly **36**(2) (2012) 649–676
5. Sen, R., Singh, S.S., Borle, S.: Open source software success: Measures and analysis. Decision Support Systems **52**(2) (2012) 364–372
6. Zhu, K.X., Zhou, Z.Z.: Research note-lock-in strategy in software competition: Open-source software vs. proprietary software. Information Systems Research **23**(2) (2012) 536–545
7. Raja, U., Tretter, M.J.: Defining and evaluating a measure of open source project survivability. Software Engineering, IEEE Transactions on **38**(1) (2012) 163–174
8. Zahra, S., Nazir, A., Khalid, A., Raana, A., Majeed, M.N.: Performing inquisitive study of pm traits desirable for project progress. International Journal of Modern Education and Computer Science (IJMECS) **6**(2) (2014) 41
9. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in github: transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, ACM (2012) 1277–1286
10. Russell, M.A.: Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More. " O'Reilly Media, Inc." (2013)