Acceptance Factors of Pull Requests in Open-Source Projects

Daricélio Moreira Soares[1,2]
[1]Federal University of Acre
Rio Branco - AC, Brazil
daricelio@ufac.br

Manoel Limeira de Lima Júnior[1,2]
[1]Federal University of Acre
Rio Branco - AC, Brazil
limeira@ufac.br

Leonardo Murta[2]
[2]Institute of Computing
Fluminense Federal University
Niterói - RJ, Brazil
leomurta@ic.uff.br

Alexandre Plastino[2]
[2]Institute of Computing
Fluminense Federal University
Niterói - RJ, Brazil
plastino@ic.uff.br

ABSTRACT

Distributed version control systems provide support for pull request strategy, which is used to register external contributions in collaborative software projects. The data present on a pull request can provide insights of factors that have influence on the acceptance or rejection of contributions in open source projects. Furthermore, the discovery of knowledge about pull requests allows confirming or denying existing hypotheses and helps software developers and project managers to guide their actions. This work proposes the use of data mining, more specifically, the extraction of association rules, to find patterns that exert influence on the acceptance (merge) of a pull request. The results suggest that: (i) the use of association rules allows to identify which factors increase the likelihood of a pull request merge; (ii) the identification of attributes that influence the merge reveals important knowledge about the pull request model; and (iii) with the use of association rules, it is possible to determine which factors contribute to a faster merge.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement; D.2.8 [**Artificial Intelligence**]: Learning

General Terms

Management, Measurement, Documentation, Experimentation

Keywords

pull request, data mining, association rules, software repositories.

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695856

1. INTRODUCTION

Version Control Systems (VCS) are one of the main strategies for configuration management, allowing developers to control changes over software artifacts through development and maintenance [3]. Recently, Distribuited Version Control Systems (DVCS) appeared, enhancing support for collaborative software development. A DVCS lets developers control multiple remote repositories, enabling the joint work of different groups simultaneously on the same project. An example of this type of system is Git, which registers the whole history of the software evolution. This way, hosting tools, such as GitHub, are characterized as a rich source of information to support the process of configuration management.

An important feature present on a DVCS is the systematization of how external team members can collaborate with the core team members [12]. External members can clone a repository, make changes to the code, and then request the changes to be merged back into the main repository. This process of requesting the changes to be merged back into the repository is done via a pull request. A pull request contains a contribution that will be accepted or denied by any member of the core team responsible for its analysis. It can be seen as a patch, but is handled directly by the DVCS.

Recent research on this topic aims at investigating the process of pull request, observing characteristics that positively or negatively influence the request acceptance. Works such as Gousios's [5], Rahman's [9], and Tsay's [10] proposed an exploratory study on the software development model based on pull requests. These works seek to understand the reasons for making a merge and, in some cases, try to foresee the merge of a pull request. The results found in these works evidenced rather interesting characteristics: (i) pull requests in projects coded in certain programming languages have a higher rate of acceptance [9]; (ii) a considerable amount of accepted pull requests are analyzed in a short timespan [5]; and (iii) socio-technical factors may also influence on the merge of a pull request [10]. However, these works did not explore the characteristics inherent to the pull requests, be them in respect to the projects, the developer, or even the contribution itself.

In this paper we adopted a data mining technique, more specifically, the extraction of association rules, in order to

identify new and useful patterns from pull requests data. Through this technique, it is possible to discover new relationships and features in the data being analyzed, providing useful information that would probably be ignored if a superficial exploratory analysis were being used instead. As a result, we aim at mining patterns which identify the features inherent to pull requests that increase or decrease the likelihood of merge. The key association rule metrics — support, confidence, and lift — suggest how powerful and reliable an existing pattern is and indicate the power the features have to increase the chances of acceptance of a pull request, that is, the incorporation of code through a merge.

Data mining techniques have been employed in the extraction of knowledge from software repositories [2, 13]. This kind of data exploration aims at evidencing useful information to aid software managers and developers in fulfilling their roles. The exploratory analysis done through the extraction of association rules focuses on the discovery of intrinsic information from data set. This new understanding may both corroborate or reject a pattern or theory established, as well as it can reveal new information on the process of collaboration in software development. It is not mandatory, in the association rules approach, to use hypotheses or research questions, since the technique finds patterns previously unknown, which are frequent in the repository and may be useful to software engineering.

The results of our research indicate that: (i) the use of association rules helps to identify which factors increase the chance of a merge given a pull request; (ii) the identification of attributes that influence the merge of a pull request provides valuable knowledge about the pull request evaluation process; and (iii) through the utilization of association rules and measurements thereof, it is feasible to verify which factors contribute to a fast merge.

This work is organized as follows. Section 2 discusses the main works related to this exploratory investigation. Section 3 presents the research methodology, the data used to perform the experiments and a conceptual review of the techniques used. Section 4 presents the results obtained and, last, Section 5 brings the conclusion of this work, highlighting its main contributions.

2. PULL REQUEST-BASED SOFTWARE DEVELOPMENT

Rather recent research has showed interest in the investigation of pull request-based Software development [5, 8, 9, 10]. Pull request is a process through which an external or internal collaborator contributes in a project stored in software repositories managed by a DVCS. GitHub, for instance, is a shared web hosting service for projects that make use of the DVCS Git and offers a mechanism of collaboration in the form of pull request. In general, studies [5, 9, 10] have been carried out aiming at comprehending the influence of factors over the acceptance of these requests and, thus, the fulfilling of the merge. Next, some of the related works on collaborative development based in external collaborations are discussed.

Gousios et al. [5] aimed at comprehending the life cycle of a pull request and analyzing the factors that contribute to its acceptance and the time required to accomplish that. This study reveals that 14% of the repositories stored on GitHub make use of the pull request strategy. Regarding the size of the contributions, the study evidences that in 90% of them, there are less than 20 files handled and a number inferior to 500 lines of code modified. Furthermore, about the life cycle of a pull request, it shows that 80% of the contributions are accepted or denied in 3 to 7 days. The work does not suggest, however, how these factors can contribute to the increase of merge rate given the pull requests.

Seeking to deepen the understanding of how to use the information contained in open-source software environments to evaluate pull requests, Tsay et al. [10] analyze the socio-technical signals associated with merging pull requests. The results suggest that contributions that include test code are more likely to be merged. The existence a social connection between the requester, the project, and the developer responsible for the analysis of the contribution, also influences merge decisions positively. A statistical analysis points out that some factors contribute to the decrease of the probability of acceptance, such as the number of modified files (many files) and the number of comments (various comments). Although it uses attributes inherent to the requester, project, and the pull request itself, the work does not explore how these features altogether affect the merge of a pull request.

Rahman and Roy [9] did a comparative study between contributions whose merge requests were accepted and the ones whose the requests were rejected, analyzing a varied collection of data, such as comments, collaboration history, project statistics, and the developer that makes the pull request. The results of this work show that the programming language in which the code of pull request is written is an important aspect to be analyzed. The authors suggest that, in contributions made in Scala, C, C#, and R, a higher merge rate can be observed. However, they do not show the degree of influence of the languages in the increase or decrease of the chance of merge. In our approach, we also analyze the influence of programming languages over the acceptance of pull requests. However, we adopt measurements to quantify such influence.

3. MATERIALS AND METHODS

Our work proposes an exploratory study on pull requests through data mining. More specifically, we extract association rules, aiming at discovering relationship among features of pull requests obtained from GitHub. In this section, we detail the materials and methods used in our experiment.

3.1 Pull Request Data

The data used in our experiments were made available thanks to the data mining challenge of the 11th Working Conference on Mining Software Repositories - MSR[1], which was part of the 36th International Conference on Software Engineering (ICSE). The knowledge base was extracted using GHTorrent [4].

The data set used in the experiment comprises of 61,592 pull requests made on 72 different projects. The attributes considered in the analysis are: project_id (project identifier); language (programming language); developer_type (main_team or external); first_pull (informs if the pull request is the first made by the requester); commits_pull (amount of commits per pull request); files_added (amount of files added); file_edited (amount of files edited); files_ removed (amount of files removed); files_changed (amount of files added, re-

[1]http://2014.msrconf.org/challenge.php#challenge_data

moved, and edited); analysis_time (time required to analyze the pull request); status_pull (pull request final status). The value of the last attribute determines whether the pull request is accepted (merged) or rejected (closed).

3.2 Association Rules

One of the main tasks in data mining is the extraction of association rules, whose objective is to find associations between the attributes of a repository [7]. An association rule represents a relationship pattern between data items of an application domain that occurs with a certain frequency in the data set.

In this work, we use the concept of multidimensional association rule. Given a database D, a multidimensional association [7] rule $X \rightarrow Y$ is an implication of the form: $X_1 \wedge X_2 \wedge \cdots \wedge X_n \rightarrow Y_1 \wedge Y_2 \wedge \cdots \wedge Y_m$, where $n \geq 1, m \geq 1, X_i (1 \leq i \leq n)$ and $Y_j (1 \leq j \leq m)$ are conditions defined on distinct attributes of D [7, 11].

A rule $X \rightarrow Y$ indicates, with a degree of certainty, that the occurrence of X (antecedent) implies the occurrence of Y (consequent). The relevance of an association rule is evaluated by two main metrics of interest: support and confidence [11]. The support metric is defined by the percentage of instances that satisfy the conditions of the antecedent and the conditions of the consequent, computed as follows: $Sup = T_{X \cup Y}/T$, where $T_{X \cup Y}$ represents the number of records that satisfy the attributes in X and Y. T is the number of records in D. The measurement of confidence represents the probability of occurrence of the consequent, given the occurrence of the antecedent. It is obtained in the following manner: $Conf = T_{X \cup Y}/T_X$ where T_X represents the number of records that satisfy the conditions of the antecedent X. Support and confidence are used as a filter in the process of mining the association rules, that is, only the rules characterized by having a minimum support and confidence (defined as an input parameter) are extracted.

Another metric of interest considered in this work is called Lift, which indicates how more frequently Y occurs given that X occurs. *Lift* is computed as the quotient of the confidence of the rule and the support of its consequent, i.e., $Lift(X \rightarrow Y) = Conf(X \rightarrow Y)/Sup(X)$. When $Lift = 1$ there is a conditional independence between X and Y, that is, the antecedent does not interfere in the occurrence of the consequent. On the other hand, $Lift > 1$ indicates a positive dependence between the antecedent and the consequent, meaning that the occurrence of X increases the chances of occurrence of Y. Conversely, when $Lift < 1$ there is a negative dependence between the antecedent and the consequent, which indicates that the occurrence of X decreases the chances of occurrence of Y. In this work, the interest metrics (support, confidence, and lift) are used to evaluate the relevance of the rules that indicate factors that affect the merge of a pull request.

The Apriori [1] algorithm was used in order to obtain the association rules. It is available in the data mining tool WEKA[2] (Waikato Environment for Knowledge Analysis) [6], which aggregates a collection of machine learning and data mining algorithms.

4. RESULTS AND DISCUSSION

[2]http://www.cs.waikato.ac.nz/ml/weka/

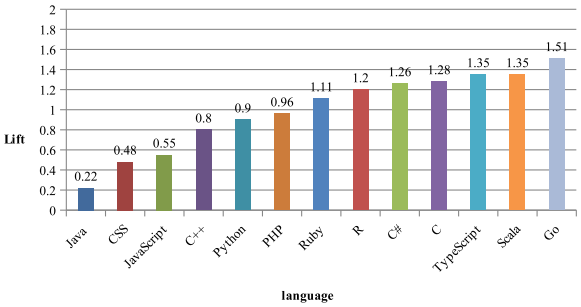

Figure 1: Lift of the rule: language → merge.

The approach proposed in this work to investigate factors related to the merge of pull requests uses extraction of association rules, aiming at discovering patterns among pull request attributes. Accordingly, the association rules presented in the experiments indicate the power that some attributes have over the occurrence of a merge of a pull request. In this section, the main results are presented and analyzed.

4.1 Influence of Attributes of the Merge of a Pull Request

The analysis of the association rules shows that some individual attributes may influence the occurrence of a merge, as well as the combination of distinct attributes.

The experiments evidence that some programming languages hold strong influence over the merge. Rahman [9] points out that there is a higher number of merge occurrences in some programming languages, but without quantifying the influence that the programming language holds over the merge. Through the lift metric of a rule of type $language \rightarrow status\_pull = merge$, which indicates how more frequently the merge becomes (consequent of the rule) given that a certain programming language occurred (antecedent of the rule), it is possible to quantify how programming languages influence the merge of a pull request. Figure 1 shows the lift of each rule $language \rightarrow status\_pull = merge$, for each of the programming languages present in the data set, indicating the increase ($Lift > 1$) or decrease ($Lift < 1$) of the probability of a merge.

Figure 1 shows that pull requests written in programming languages like Java, CSS, JavaScript, and C++, have the merge chances reduced, whereas C#, C, TypeScript (few projects), Scala, and Go (few projects) increase the chances of occurrence of a merge. The association rules make it feasible to perceive not just the increase/decrease of the chances of merge, but also quantifying this variation. By analyzing the lift of the rule $Java \rightarrow status\_pull = merge$, it is possible to verify that contributions made in Java have 78% less chance of merge. Similarly, a pull request coded in JavaScript has 45% less probability of merge. Conversely, a contribution written in C# has a chance 26% higher of being incorporated. The knowledge extracted with the association rules involving the programming language attribute may be useful to realize that the pull requests of certain projects can have more/less merge changes.

Another characteristic that affects the merge is related to the size of the pull request, represented by the attribute

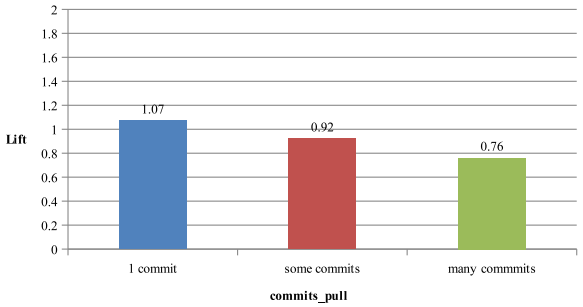

Figure 2: Lift of the rule: commits→merge.

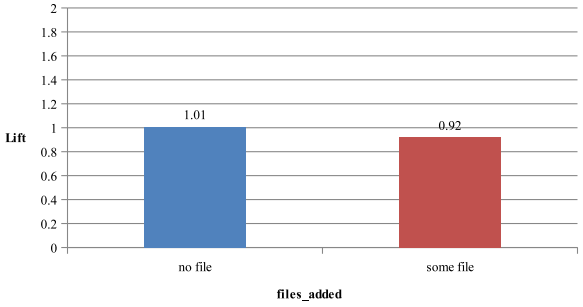

Figure 3: Lift of the rule: files_added→merge.

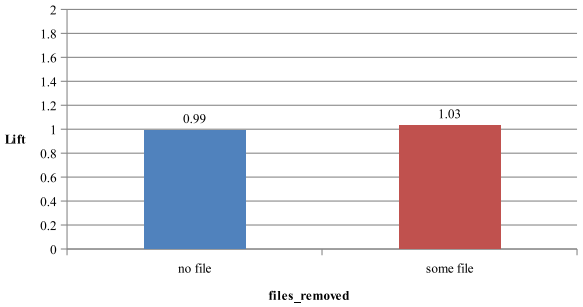

Figure 4: Lift of the rule: files_removed→merge.

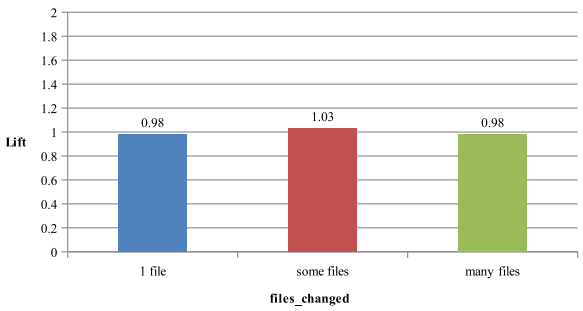

Figure 5: Lift of the rule: files_changed→merge.

that indicates the number of commits. Figure 2 shows the values of the lift metric for rules of type $commits\_pull$[3] $\rightarrow$ $status\_pull = merge$ and enables the observation that the greater the contribution (in terms of number of commits), the smaller the chances of merge. This suggests that the amount of commits heighten the complexity of the analysis.

We also analyzed the values of lift for rules involving the files_changed attribute (amount of files modified in a pull request) in three different dimensions: amount of files added, amount of files removed, and amount of files changed (summation of the files added, removed, and edited in a pull request, respectively). The results suggest that when a pull request does not add files (Figure 3), this attribute does not hold any influence ($lift = 1.01$) over the merge and when there are added files, the chances of merge are reduced in 8%. When there is removal of files (Figure 4) in a pull request, there is practically no influence, since the values of lift tend to be next to 1. The total number of files changed (Figure 5) in a pull request presents, individually, similar behavior as the files removed.

Other rules illustrate the effect of two other attributes over the merge: developer type (main_team or external collaborator) and if the collaboration submitted is the first that the requester made (first_pull assuming values true or false). The results demonstrate that pull requests of external collaborators (Figure 6) have 13% less chance of being accepted, while contributions made by members of the main team increase in 35% the probability of merge. On the other hand, when the pull request is the first made by a developer, the chance of merge is decreased in 32% (Figure 7). Furthermore, the results show that, when the collaborator has already submitted a pull request before, his or her contribution has 9% more chance of being merged.

[3]"some commits" represents 2 to 4 commits; "many commits" represents 5 or more commits.

4.2 Combination of Attributes Influencing the Merge of a Pull Request

The results presented so far are relative to rules composed of only one attribute in the antecedent. We could observe that some attributes characterize a more complex pull request and reduce the changes of merge: multiple commits, files added, external developer, and first time. This opens a question related to it the accept of pull request is influenced by the combination of such attributed. Table 1 shows some rules with more than one attribute in the antecedent. The consequent of all the rules presented have status_pull=merge as their value.

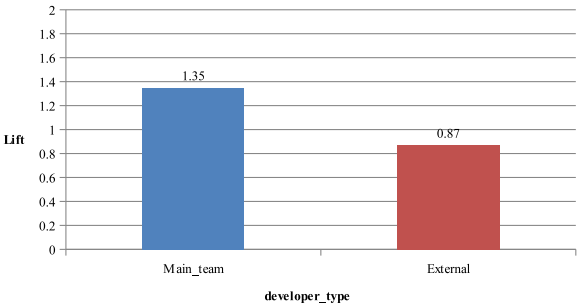

Figure 6: Lift of the rule: developer_type→merge.

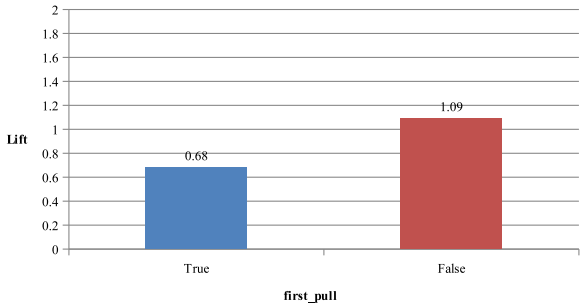

Figure 7: Lift of the rule: first_pull→merge.

The confidence of Rule 1 indicates that requests written in Scala that have only one commit are merged in 77% of the times. The lift of this rule indicates that these two conditions increase the chances of merge in 43%. It is important to observe that considering only the programming language (Scala — see Figure 1), the increase in the chances of merge was 35%. So this combination of the number of commits and the programming language revealed an important pattern. Rule 2 demonstrates other situation where the number of commits increases the chances of merge of a pull request of a certain programming language. The rule shows that pull requests written in C# and that have only one commit have their probability of merge increased in 31%, whereas the lift associated with the language C# in relation to the merge is only 1.26.

Rule 3 (Table 1) displays a conjunction of good factors with positive influence over the acceptance of a pull request. It is noteworthy that a request with a sole commit, that does not add new files, made by a member of the main team which had already made a pull request before, has the chance of merge increased in 40%. The rules are also able to evidence that pull requests of certain projects reveal specific patterns. In rule 4, it is possible to verify that a pull request made in project Akka composed by only 1 commit is accepted in 92% of the times. In addition, this antecedent increases by 70% the chances of merge. It is worth mentioning that this project has, on its own, $lift = 1.63$. Rule 5 reveals another combination of attributes that increases the probability of merge. In this scenario, it is notable that requests made in Python, by members of the main team with some experience with pull requests, have 57% more chance of being accepted.

4.3 Factors that Influence the Fast Acceptance or Rejection of a Pull Request

Useful patterns are mined when considering the attribute that indicates the time necessary to analyze a pull request. Figure 8 illustrates the values of lift for the rules that contain the attribute analysis_time[4] in their antecedent and merge in their consequent, and displays how more frequently merge occurs in relation to the time taken to analyze the contribution. It is possible to observe that the increase in the time needed to analyze a pull request reduces the chances of acceptance of such request.

In a complementary analysis, shown in Table 2, rules with

[4]Very Fast (until 24 hours); Fast (24 to 72 hours); Medium (72 to 168 hours); Slow (160 to 240 hours); Very Slow (more than 240 hours).

Table 1: Relevant association rules and their measures of interest.

#	Antecedent	Sup (%)	Conf (%)	Lift
1	language = Scala ∧ commits_pull = 1 commit	4.3	77	1.43
2	language = C# ∧ commits_pull = 1 commit	2.0	71	1.31
3	commits_pull = 1 commit ∧ files_added = no file ∧ developer_type = main_team ∧ first_pull = false	10.5	76	1.40
4	project = Akka ∧ commits_pull = 1 commit	1.5	92	1.70
5	language = Python ∧ developer_type = main_team ∧ first_pull = false	1.0	85	1.57

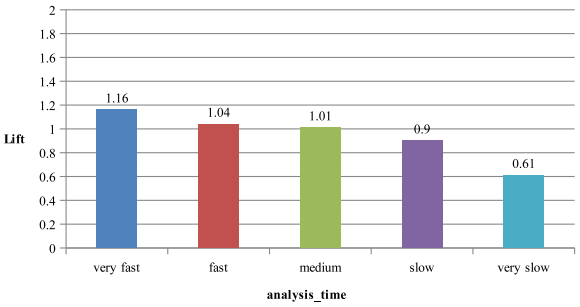

Figure 8: Lift of the rule: analysis_time→merge.

the attribute analysis_time in the consequent were mined aiming at identifying attributes that influence the acceptance time. Rule 1 indicates that requests with one commit have chances increased in 17% of being accepted rather quickly, that is, in the span of 24 hours. Similarly, requests with only one commit and written in Scala have their chances of fast merge (from 24 to 72 hours) increased by 116%.

Other results also suggest the discovery of rules that evidence factors that influence the fast rejection of a pull request. Rule 3 (Table 2), for instance, shows that pull requests with many commits, performed for the first time by an external developer, are rejected in 24 hours in 25% of the times. In addition, this rule indicates that the occurrence of the antecedent increases in 32% the chance a very fast rejection.

5. CONCLUSIONS

The results presented in this work corroborate that association rules mining enables the discovery of useful knowledge to developers and project managers. The association rules allow the perception of characteristics that increase the chances of occurrence of the merge of a pull request, revealing a behavior that, in some cases, can be followed by a developer that seeks to ensure that the merge of his or her pull request occurs. The technique makes it possible to

Table 2: Association rules involving analysis_time and their measures of interest.

#	Antecedent	Consequent	Sup (%)	Conf (%)	Lift
1	commits_pull = 1 commit	analysis_time = very fast ∧ status_merge = merge	24.1	37	1.17
2	language = Scala ∧ commits_pull = 1 commitit	analysis_time = fast ∧ status_merge = merge	1	17	2.16
3	commits_pull = many commits ∧ developer_type = external ∧ first_pull = true	analysis_time = very fast ∧ status_merge = closed	1.9	25	1.32

discover patterns that grant the project manager a further comprehension of the features of pull requests submitted to the repository.

Mining of patterns in the pull requests provides comprehension of factors that influence the increase or decrease of chances of merge. We could observe that the following factors have influence in the merge of pull requests: programming language, number of commits, files added, external developer, and first pull request. We could also observe that these factors, when combined, may improve the results. Moreover, we note that these factors also impose influence on how fast pull requests are merged or rejection. Finally, we could observe that the speed of merge is also correlated to the acceptance of the pull request. The knowledge obtained in this research allowed the identification of situations in which the pull requests are likely to be accepted/closed and the time required to accomplish that.

As a future work, we intend to carry out additional experiments over a larger number of software repositories. Furthermore, another exploration perspective is the utilization of attributes derived from the source code, for example: changed classes, changes files (contents of files) and software metrics.

6. ACKNOWLEDGMENTS

We would like to thank CNPq, CAPES and FAPERJ for the financial support.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Y. C. Cavalcanti, P. A. Mota Silveira Neto, I. C. Machado, T. F. Vale, E. S. Almeida, and S. R. d. L. Meira. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal Software Evolution and Process.*, 26(7):620–653, July 2014.

[3] J. Estublier. Software configuration management: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE, pages 279–289, New York, NY, USA, 2000. ACM.

[4] G. Gousios. The GHTorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR, pages 233–236, Piscataway, NJ, USA, 2013. IEEE Press.

[5] G. Gousios, M. Pinzger, and A. V. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355, New York, NY, USA, 2014. ACM.

[6] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[7] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques: Concepts and Techniques*. Elsevier, June 2011.

[8] R. Padhye, S. Mani, and V. S. Sinha. A study of external community contribution to open-source projects on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR, pages 332–335, New York, NY, USA, 2014. ACM.

[9] M. M. Rahman and C. K. Roy. An insight into the pull requests of GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR, pages 364–367, New York, NY, USA, 2014. ACM.

[10] J. Tsay, L. Dabbish, and J. Herbsleb. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE, pages 356–366, New York, NY, USA, 2014. ACM.

[11] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier, Feb. 2011.

[12] M. Zandstra. Version control with git. In *PHP Obejcts, Patterns, and Practice*, pages 365–382. Apress, Jan. 2013.

[13] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.