UNIVERSIDADE FEDERAL FLUMINENSE

DARICÉLIO MOREIRA SOARES

ON THE NATURE OF PULL REQUESTS: A STUDY ABOUT THIS COLLABORATION PARADIGM OVER OPEN-SOURCE PROJECTS USING ASSOCIATION RULES

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

DARICÉLIO MOREIRA SOARES

ON THE NATURE OF PULL REQUESTS: A STUDY ABOUT THIS COLLABORATION PARADIGM OVER OPEN-SOURCE PROJECTS USING ASSOCIATION RULES

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense, in partial fulfillment of the requirements for the degree of Doctor of Science. Área: Systems and Information Engineering.

Advisor:
LEONARDO GRESTA PAULINO MURTA

Co-advisor:
ALEXANDRE PLASTINO DE CARVALHO

NITERÓI

2017

DARICÉLIO MOREIRA SOARES

ON THE NATURE OF PULL REQUESTS: A STUDY ABOUT THIS COLLABORATION PARADIGM OVER OPEN-SOURCE PROJECTS USING ASSOCIATION RULES

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense, in partial fulfillment of the requirements for the degree of Doctor of Science. Área: Systems and Information Engineering.

Approved in June of 2017.

Niterói

2017

My flesh and my mind may fail, but God is my strength and my portion forever.

(Salms 73:26)

The burden is proportional to the forces, as the reward is proportional to resignation and courage.

(Allan Kardec)

I dedicate this work to God, to the women of my life, Ligiane and Célia, and to my new source of inspiration, my son Davi.

Acknowledgement

First and foremost, I would like to thank God, who with His everlasting goodness, guides us through the paths of life and allows us to overcome obstacles.

I would like to thank my wife Ligiane. She has been my foundation. Thank you for your dedication and love. I also thank you for your patience and constant motivation.

I would also like to thank my mother Célia. Her struggle, grip, and dedication will always serve as examples to follow throughout my life. Thank you for your unconditional love.

I thank my family and friends for having cheered for my success throughout this journey.

I thank my advisors Leonardo Murta and Alexandre Plastino for all their teachings, technical support, motivation, and demonstrated examples. It has been an honor to work with you.

I additionally thank my fellow professors from the Informatics field at Universidade Federal do Acre for their camaraderie, and professor Sérgio Brazil for his continued support.

I especially thank my friends Manoel Junior and Laura Sarkis for their involvement, indispensable help, and advice offered in moments of distress.

Last but not least, I thank UFAC, UFF, and CAPES for the financial support provided over the course of my doctor's degree.

Resumo

O desenvolvimento de software livre está comumente inserido num contexto distribuído e colaborativo, o que permite o recebimento de contribuições externas. Um paradigma emergente empregado para a sistematização dessas contribuições é denominado *pull request*. Nesse paradigma, colaboradores externos que desejam contribuir com um projeto criam um *fork* a partir do repositório do projeto, fazem suas alterações e enviam um *pull request* à equipe principal, que revisará a contribuição e decidirá sobre a incorporação ou não ao repositório. *Pull requests* podem conter correção de *bugs*, refatoração de código ou adição de novas funcionalidades, por exemplo. Atualmente, poucas informações sobre a natureza dos *pull requests* são conhecidas no cenário de projetos de software livre. Alguns trabalhos investigaram características de *pull requests* relacionadas à aceitação, tempo de vida e atribuição de revisores. No entanto, esses estudos negligenciam a exploração de questões importantes e que estão em aberto. Nesta tese, realizamos um conjunto de estudos baseados na extração de regras de associação sobre 88 projetos de software livre, detalhando a natureza dos seus 132.660 *pull requests* por meio da: (1) identificação de padrões frequentes a partir de milhares de *pull requests*; (2) avaliação da extensão e força desses padrões; e (3) análise qualitativa que explica a ocorrência de alguns deles. Nossos resultados indicam que as características físicas dos *pull requests*, o perfil dos colaboradores, aspectos sociais do processo e a localização das contribuições são fatores que influenciam, em diferentes intensidades de força, na aceitação/rejeição, no tempo de vida e na atribuição de revisores. A identificação desses padrões pode apoiar desenvolvedores e gerentes de projeto na compreensão da natureza dos *pull requests* e guiá-los a práticas que permitam extrair ao máximo os benefícios desse paradigma de colaboração.

Palavras-chave: *Pull Request*, Regras de Associação, Aceitação, Tempo de Vida, Atribuição de Revisores.

Abstract

Open-source software development is nowadays inserted in a distributed and collaborative context, which enables to receive external contributions. An emerging paradigm employed for the systematization of these contributions is named pull request. According to this paradigm, external developers wishing to contribute to a project fork the project repository, make their changes, and send a pull request to the project's core team, who will review the contribution and decide whether or not to integrate it into the repository. Pull requests may contain bug fixes, code refactorings, or new features, for example. Currently, few information about the nature of pull requests is known in the scenario of open-source projects. Some work investigated pull requests characteristics related to acceptance, lifetime, and reviewers assignment. However, all of these studies neglected the exploration of certain aspects that are still open. In this thesis we performed a set of studies based on the extraction of association rules from 88 open-source projects, detailing the nature of their 132,660 pull requests through: (1) the identification of frequent patterns from thousands of pull requests; (2) the assessment of the extent and strength of these patterns; and (3) a qualitative analysis that explains the occurrence of some patterns. Our results indicate that physical characteristics of the pull requests, collaborators profile, social aspects of the process, and location of contributions are factors that influence, in different intensities, on: the acceptance/rejection, lifetime, and reviewers assignment. The identification of these patterns can support developers and project managers in understanding the nature of pull requests and guide them to practices that maximize the benefits of this collaboration paradigm.

Keywords: Pull Request, Association Rules, Acceptance, Lifetime, Reviewers Assignment.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The software development process configures a scenario wherein the work effort takes place in a collaborative and distributed manner, with certain regularity [2, 3]. Open-source projects represent a typical instance of this scenario usually with large development teams. Often the team members work from different parts of the world, but in consonance for the creation of a software product [2]. In spite of the difficulties imposed by the distance, various open-source projects have successfully achieved the production of large and complex systems [2].

In open-source projects, several people can contribute to the development of software. For instance, an external developer might discover a flaw in the source code, fix it, and then request the incorporation of the change into the software. The systematization of such contribution process is of great importance.

An emerging paradigm employed for the systematization of contributions in open-source projects is named pull request [1]. According to this paradigm, external contributors and core team members can make isolated modifications in artifacts and then request the integration of their modifications into the project main repository. For instance, a pull request may contain bug fixes, code refactorings, or new functionalities, which, after being analyzed by a core team member, will be either accepted (i.e., incorporated) or rejected (i.e., closed) [4, 5]. In summary, when contributing to a project, the developer must fork the remote repository as he or she does not have direct write (i.e., push) access to it. Next, a copy (i.e., clone) must be made from the fork repository to the developer's computer, wherein he or she will be able to perform modifications (i.e., commits) in the software artifacts and sent (i.e., push) the changes to the forked repository. Next, he or she requests the changes to be merged into the main repository, by filing a pull request. Following its submission, the pull request must undergo a review process carried out by one or more core team members [1].

1.1 Motivation

Comprehending the factors associated with the acceptance and rejection of pull requests is relevant, following the premise that both the core team and requesters expect contributions to be accepted, if they are viable and useful to the project. Besides, pull requests can also aid in systematizing the integration of code produced by core team members, forcing other core team members to review and discuss about the contribution prior to incorporating it into the repository. Within this scenario of internal contribution, understanding the factors that influence the acceptance and rejection of pull requests is also important.

The submission of a pull request marks the commencement of its lifetime. During review, core team developers may comment and discuss around the pull request, potentially demanding additional modifications to the requester. This interaction may impact the necessary time to evaluate the contribution, and may also, as a result, impact the final status of the pull request (i.e., acceptance or rejection). In this way, understanding factors that influence the lifetime of pull requests is important to understand their nature.

Another important issue in the pull request scenario is the assignment of a reviewer to analyze it. The decision to review a given pull request may come directly from the core team members with interest on the pull request, potentially incurring in inappropriate assignments. Comprehending the nature of reviewers' allocation for analyzing pull requests is an important step to better use resources, avoid conflicts of interest among requesters and reviewers, and leverage the workforce provided by external contributors.

Better understanding the four aforementioned characteristics of pull requests (acceptance, rejection, lifetime, and assignment) helps both the requesters and the core team. Requesters, who file pull requests in open-source projects, would benefit from understanding the factors that may be of influence to the acceptance of pull requests, since this knowledge may aid them in maximizing the chances of acceptance of their contributions. For example, by realizing that editing few lines of code in a pull request can contribute to acceptance, a requester could guide his or her actions in order to partition changes into scarcer batches. Besides, this would enable the requester to be more efficient and productive, not wasting time sending pull requests that present characteristics that might influence their acceptance in a negative manner. It is important to note that, although software teams may have specific culture and habits for their collaborative processes, replicating past patterns may not always be appropriate. Nonetheless, the discovery of such patterns may motivate process improvement.

Lately, several studies have been carried out seeking to improve the understanding of how pull requests are adopted in open-source projects. The works of Gousios et al. [4], Tsay et al. [6], Rahman and Roy [7], and Padhye et al. [8], for instance, aimed at characterizing the pull-based development paradigm and raised questions on the acceptance of contributions. The works of Gousios et al. [4] and Yu et al. [9] investigated factors related to pull request lifetime. Yu et al. [10], Limeira et al. [11], Jiang et al. [12], Ying et al. [13], and Rahman et al. [14] explored recommendation strategies for pull request reviewers. Nonetheless, all of these studies neglected the exploration of certain important matters, such as: the identification of joint characteristics that influence on these scenarios; the extent and strength of this influence; and the explanation of this influence.

1.2 Goals

This thesis presents a set of studies on the nature of the pull-based development paradigm. The studies comprise the identification of factors that have influence on pull request acceptance/rejection, lifetime, and reviewer assignment.

The studies aim at answering the following general research questions. Each of these general research questions is further decomposed into specific research question in the respective chapter.

- Which factors influence the acceptance of pull requests in open-source projects? (Chapter 3)
- Which factors influence the rejection of pull requests sent by core team members in open-source projects with high acceptance rates? (Chapter 4)
- Which factors influence on lifetime of pull requests in open-source projects? (Chapter 5)
- Which factors influence the assignment of reviewers for pull requests in open-source projects? (Chapter 6)

We adopted a data mining technique named association rule extraction to answer these research questions. Through this technique, we explored 132,660 pull requests data from 88 software repositories. The extraction of association rules configures an efficient technique to identify significant correlations, frequent patterns, associations, or causal structures between items in a dataset [15]. Besides, it provides useful information that would probably be ignored if a superficial exploratory analysis were being used instead. In summary, an association rule represents a relationship pattern between data items that occurs with a certain frequency. Moreover, it has measures that are able to indicate the relevance, validity, and strength of such patterns [16]. We used association rules to allow the identification of relationships between data of different dimensions, such as the physical, social, and metadata characteristics of pull requests. This technique was chosen because it allows identifying relationships between attributes of a dataset, based on objective metrics that indicate their relevance. Thus, association rules differ from the classical statistical analysis, such as regressions, that would hardly find out hidden patterns to could answer our research questions. In addition, association rules extraction counts on efficient algorithms, available in several tools that allow a deep exploratory analysis. Association rules allow obtaining unknown patterns and not only verify hypotheses related to the application domain. Finally, the rules are formed by conditions involving attributes and their values in the antecedent and the consequent, which allows an analysis of the data at finer grain This differs from obtaining correlations between variables (attributes).

1.3 Research Design and History

Initially, for the development of the studies, we reviewed the literature considering works that explored the same scenarios of this thesis. To do so, we used as start set the work of Gousios et al. [4], which has a large number of citations and presents an overview of the collaboration paradigm. This start set was employed in a backward and forward snowballing process.

In addition, even though each study explores different scenarios in the context of pull requests, and specific analyses are necessary for the comprehension of the findings in each case, there are some commonalities among them. Specifically, all four studies employed the Knowledge Discovery in Databases (KDD) process, as followesd [17]: (1) data selection; (2) preprocessing; (3) transformation and data enrichment; (4) association rules extraction; and (5) results interpretation and evaluation.

For data selection (step 1), we used the GHTorrent[1] tool [18], which extracts information from versioned repositories stored on GitHub. For preprocessing (step 2), we performed data cleaning and attribute selection in order to remove items with missing, incorrect, or inconsistent data. In this step, in order to transform numerical attributes into discrete ones, we use a unsupervised discretization based on an instance filter available in the Weka tool. Initially, we used the discretization by simple binning, ignoring the class attribute to be defined, but considering frequency distribution. Then, we adjusted the labels for better interpretation of the extracted patterns. All labels have statistical representation in the database. For transformation and data enrichment (step 3), we discretized some numerical attributes in order to promote an improvement to the semantics of the results. For association rules extraction (step 4), we used the Apriori algorithm [19] implementation in the Weka[2] tool (Waikato Environment for Knowledge Analysis). In this step, for one each study, we extracted rules with a minimum *support* to ensure that the rules were not obtained by chance (this threshold is specified in Chapters 3, 4, 5, and 6). In addition, we have defined a minimum confidence of 5% in all studies. We use these values to get as many rules as possible about the nature of the pull request paradigm. Finally, for results interpretation and evaluation (step 5), we performed specific analyses for the comprehension and evaluation of the results obtained in each study. In order to make this stage viable we used the WekaPar plugin [20]. This plugin allows selecting rules using regular expressions, being possible to search for specific rules through the definition of target attributes. In addition, it enables the ordering of rules using measures of interest. The datasets and the framework description used in the thesis are available at GitHub[3]. Specific methodological procedures are presented in Chapters 3, 4, 5, and 6. These specific procedures are necessary because some complementary analyses were adopted in some studies, as detailed in the respective chapter.

Initially, we carried out a study on the characterization and acceptance of pull requests, wherein influence factors for this scenario were found. This study resulted in an article entitled *Acceptance Factors of Pull Requests in Open-Source Projects*[21], published at the 30th ACM/SIGAPP – Symposium On Applied Computing (SAC'15), Salamanca, Spain, 2015.

In [21], we identified that pull requests sent by core team members have greater chances of acceptance, so we conducted a study to deepen the understanding of the

[1]http://ghtorrent.org/
[2]http://www.cs.waikato.ac.nz/ml/weka/
[3]https://github.com/gems-uff/pullrequest-nature

factors that lead to the rejection of pull requests with such feature. This study resulted in a paper entitled *Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates*, published at the 14th IEEE – International Conference on Machine Learning and Applications (ICMLA'15), Miami, USA, 2015.

In [21], we preliminarily identified that lifetime can, to some extent, influence the acceptance of pull requests. Taking into consideration this aspect of pull requests, we carried out a study to identify the root causes of lifetime. This study comprises association rules extraction and qualitative analysis of the discovered patterns, which indicate factors that influence the pull request lifetime. The results of this study led to a paper entitled *Why do some pull requests take so long to be merged?*, submitted to a Journal.

Lastly, we investigated the reviewer assignment scenario. In pull-based development, a reviewer is responsible for evaluating contributions. Consequently, it is important to understand the nature of the collaboration paradigm from this perspective. This study aimed at identifying pull request features that exert influence on the assignment of reviewers to pull requests. This study resulted in a paper entitled *What Factors Influence the Reviewer Assignment to Pull Requests?*, submitted to a journal.

1.4 Organization

This thesis is organized in other six chapters, besides this introduction. Chapter 2 presents some background concepts, such as pull-based development and association rules. Chapter 3 presents the study on the pull request acceptance. Chapter 4 presents the study on the rejection of pull requests submitted by core team members. Chapter 5 presents the study on factors that influence the pull request lifetime. Chapter 6 presents the study regarding factors that influence the reviewer assignment to pull requests. Finally, Chapter 7 presents the conclusions of this thesis, as well as suggestions for future work.

Chapter 2

Background on Pull Requests and Association Rules

2.1 Introduction

Software Configuration Management (SCM) is a branch of Software Engineering concerned with managing the evolution of software products in a controlled manner, aiming at improving the efficiency of the development process and contributing to quality and time constraints [22]. SCM has tools and techniques that aid developers in keeping control of changes performed throughout the software development [23].

One of the most important SCM tools are Version Control Systems (VCS), which keep control of changes and combine procedures to manage versions of software artifacts (source code, documentation, models, etc.) created during the software development process [22, 24]. VCS provide software repositories that store information on software development/evolution and allow the creation of a new version from preexisting ones [24]. The use of repositories enables collaborative and parallel software development, even if developers are geographically dispersed, allowing retrieving artifacts, making modifications, and sharing them with other developers within the project.

A VCS can be classified as centralized (CVCS) or distributed (DVCS). CVCS employ a client-server topology characterized by multiple workspaces (the areas where developers store files and can change them) accessing a single central repository. On the other hand, DVCS employ a peer-to-peer topology characterized by the existence of several repositories in its architecture. An overview of the CVCS and DVCS operations can be seen in Figure2.1.

CVCS support three main operations: *check-out*, *check-in/commit*, and *update*. The

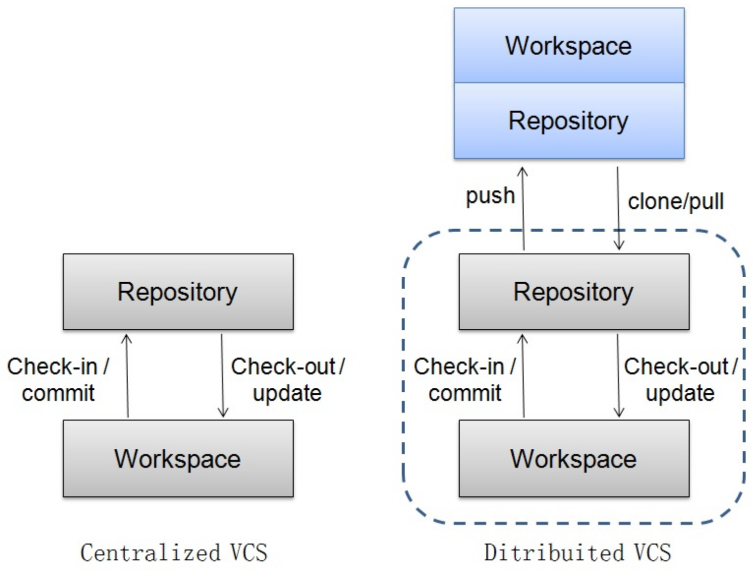


Figure 2.1: An overview of CVCS and DVCS architectures – adapted from [1].

checkout operation obtains artifacts from the main repository to the workspace. The *check-in/commit* operation represents the action of sending changes from the workspace to the main repository. Commits contain a set of changed files (more precisely, the modified lines of each file), a timestamp, the author identification, a descriptive message, and references to previous commits. The *update* operation copies parallel changes from the remote repository to the workspace [25].

Recently, DVCS emerged enhancing support for collaborative software development. DVCS allow multiple repositories in the same project, exploiting new forms of collaboration. An example of this type of system is Git[1]. In DVCS, the work mechanisms are similar to the ones of CVCS. The main difference is the existence of the *clone*, *push* and *pull* operations. The *clone* operation makes a complete copy of a remote repository into the workspace, replicating all its history. After performing local commits, the *push* operation sends all local commits to the remote repository. On the other hand, the *pull* operation copies commits from the remote repository to the local repository [1].

The data stored in software repositories represent a significant source of information and may be of aid for the understanding of factors associated with development and evolution of software systems. Recent developments in the field of software repository mining illustrate the potential to provide assistance to the decision-making process in the context of software engineering [26]. Data mining techniques have been employed for the extraction of knowledge from software repositories [27, 28] aiming at identifying useful

[1]https://git-scm.com/

information to aid project managers and developers in the fulfillment of their roles.

In open-source projects, several developers are interested in following the software project's evolution, while others are also interested in contributing with modifications. Managing such contributions in large projects, for instance, is not a trivial task. Theoretically, the larger the popularity of the software is, the greater the interest in it by the community. An emerging collaboration paradigm raised in the context of DVCS is pull request, wherein a developer forks (i.e., creates a server-side copy) of the software repository, modifies some artifacts of his or her interest, and later requests the incorporation of his or her modifications into the original repository. Although supporting pull requests is not a native feature of DVCS, multiple hosting services, such as GitHub, Bitbucket, and GitLab, support this collaboration paradigm.

Section 2.2 discusses the pull-based paradigm. Section 2.3 presents the data mining technique of rules extraction and the interest measures used to evaluate the relevance, validity, and strength of the mined rules from software repositories. Lastly, we conclude this chapter by presenting some final considerations in Section 2.4.

2.2 Pull requests

In open-source projects, the systematization of the contribution process is utterly important, since the developers interested in the projects (organized in communities) can send their contributions to the main repositories of such projects. For instance, an external developer might discover a flaw in the source code, correct it, and then request the merge of his or her changes into the main repository, thus propagating the correction.

An important characteristic of distributed and collaborative software development is the form on how external team members collaborate with the core team members [29]. Open-source software development is nowadays inserted into a collaborative and distributed context, which enables development teams to receive external contributions [2]. The reasons that motivate external developers to collaborate with a certain project include reputation-gaining, self-development (improvement of programming skills), altruism, reciprocity, and even fun [30]. They can contribute by coding, documenting, testing, or performing any other task inherent to the development of open-source projects.

An emerging paradigm employed for the systematization of contributions in open-source projects is named pull request [1]. According to this paradigm, external contributors can make isolated modifications in artifacts and then request the integration of their modifications back to the project main repository. A pull request containing either bug fixes or new features, after a review carried out by a member of the core team, may be accepted and incorporated into the project main repository or rejected.

A pull request contains a set of changes to software artifacts, which may be incorporated into the main repository (receiving an "accepted" status) or rejected (receiving a "rejected" status) [1]. This model of contribution is based on a workflow composed of seven steps [1]: (1) Forking the main repository; (2) Cloning the forked repository; 3) Changing the artifacts via commits; (4) Pushing the changes to the forked repository; (5) Requesting the incorporation of the changes into the main repository by filing a pull request; (6) Analyzing and discussing about the changes; and (7) Accepting or rejecting the pull request.

Figure 2.2 illustrates the pull-based development model. In the initial step of this process, the developer who wants to contribute to the project (called requester from now on) must fork the original repository. Forking a repository consists in cloning it in the server. In other words, the project history is copied into a new repository in the server, owned by the developer. The fork is necessary because external developers do not have push privileges in the main repository. In the second step, the requester clones the forked repository to his or her computer. This way, the requester (external or internal collaborator) now attains read and write permissions within the recently created repository copy. In the third step, the modifications made in software artifacts are sent to the recently created repository through commits. In the fourth step, changes are sent back to the forked repository. This occurs by means of a push of local commits to the fork repository on the server. Next, in the fifth step, the requester opens a pull request, asking the integration of his or her modifications into the main repository. Thus, the developers who are responsible for the evaluation of the pull request (called reviewers from now on) are notified about the pull request and begin analyzing it. At this point, in the sixth step, developers might start a discussion forum to deliberate about the contribution and the viability of integrating it. Lastly, either the pull request is accepted or closed.

A pull request features several attributes of varied dimensions, such as social and physical aspects of a contribution. When analyzing a pull request, the reviewer has access to a considerable amount of information, such as: title and description, requester identification, number of commits, number of modified files, number of lines of code modified, and the content of changes. Hence, besides objectively sending a series of changes, a pull request represents an abundant source of information on the context of collaborative

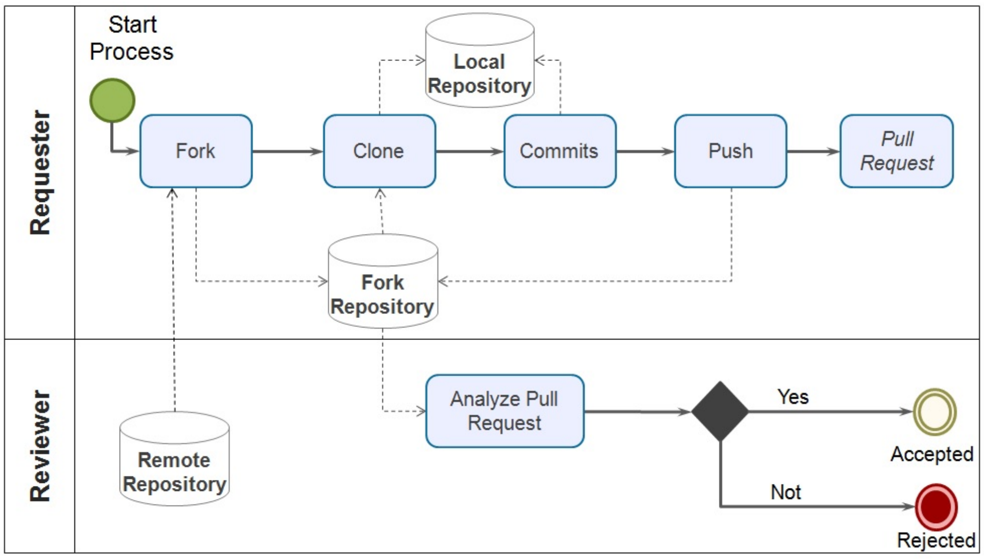
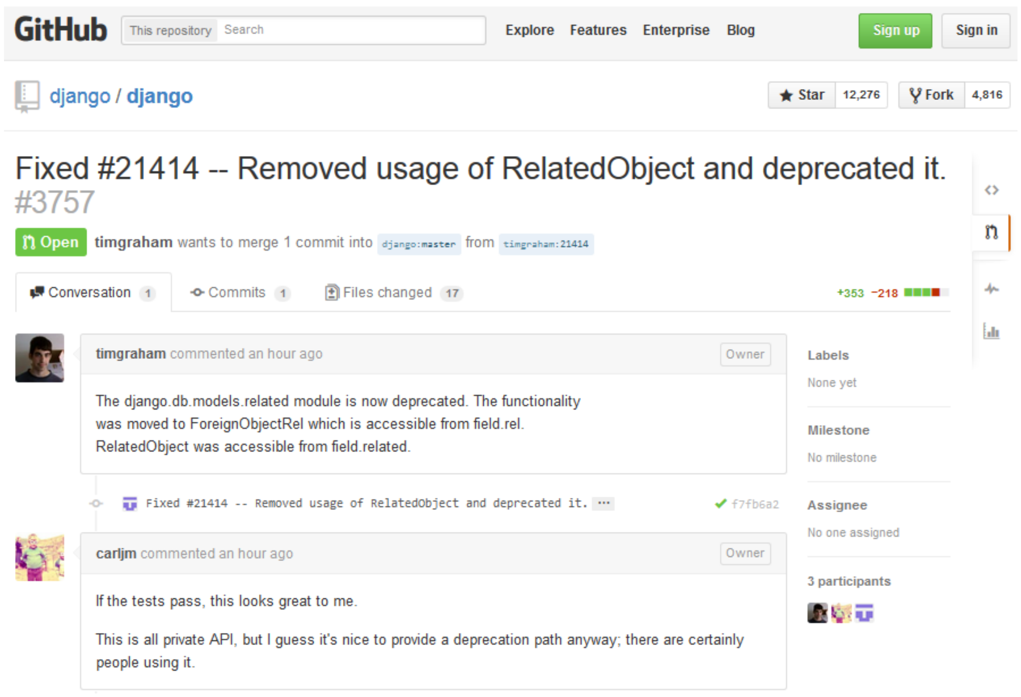


Figure 2.2: The pull request process.

software development. Much of this data can only be observed through software repository hosting tools, such as GitHub and Bitbucket, because they offer communication and social relationship functionalities among developers. For example, one can identify the popularity of a project and of a developer (number of watchers and followers), the participants involved in the discussion of a pull request, and the content of the comments of a given contribution, etc. Figure 2.3 displays a screenshot exemplifying a discussion regarding a pull request.

Even though many pull request characteristics are known at the time of its opening, some can only be observed after its submission. Characteristics such as the pull request lifetime, the final status of the contribution (accepted vs. rejected), and the developer responsible for reviewing it are only known after submission. Comprehending the factors that contribute to these characteristics is important to the understanding of the paradigm, unveiling useful knowledge for developers and project managers.

All data used in the studies presented in the next chapters were extracted by the GHTorrent tool, which analyze Git repositories hosted in GitHub. Thus, we were able to collect a large amount of data present in the repositories and, from there, derive different attributes about pull requests.



Fixed #21414 -- Removed usage of RelatedObject and deprecated it. #3757

Open timgraham wants to merge 1 commit into django:master from timgraham:21414

timgraham commented an hour ago Owner

The django.db.models.related module is now deprecated. The functionality was moved to ForeignObjectRel which is accessible from field.rel. RelatedObject was accessible from field.related.

Fixed #21414 -- Removed usage of RelatedObject and deprecated it. … ✓ f7fb6a2

carljm commented an hour ago Owner

If the tests pass, this looks great to me.

This is all private API, but I guess it's nice to provide a deprecation path anyway; there are certainly people using it.

Labels
None yet

Milestone
No milestone

Assignee
No one assigned

3 participants

Figure 2.3: GitHub pull request discussion screen.

2.3 Association Rules

The knowledge discovery in software repositories is motivated by the vast volume of stored data, which represents a scenario full of opportunities for the utilization of data mining techniques. Data mining is the main step of the Knowledge Discovery in Databases (KDD) process, which consists in discovering new and useful information and knowledge in terms of rules, models, and patterns, from large datasets. The use of data mining allows the analysis of trends and patterns extracted from history data, so as to predict future actions and aid in the decision-making process [16, 31].

The extraction of association rules is an important task in data mining, whose goal is to find relationships among data elements in a dataset [16]. An association rule represents a relational pattern between data items in the application domain that happens with a specific frequency. The extraction of association rules is a technique that allows the identification of patterns that will later be evaluated. The data mining technique adopted in this work uses the concept of multidimensional association rules [16]. Given a database D, a multidimensional association 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 $n \geq 1, m \geq 1$, and $X_i (1 \leq i \leq n)$ as well as $Y_j (1 \leq j \leq m)$ are conditions defined in terms of distinct attributes of D [16, 31]. The rule $X \rightarrow Y$ indicates, with certain degree of assurance, that the occurrence of the antecedent X implies the occurrence of the consequent Y. The relevance and validity of an association rule is evaluated by two main measures of interest: *support* and *confidence* [31].

The support measure is defined by the percentage of instances that satisfy the conditions of the antecedent and the conditions of the consequent, computed as follows: $Sup_{(X \rightarrow Y)} = T_{(X \cup Y)}/T$, where $T_{(X \cup Y)}$ represents the number of records that satisfy the conditions in X and the conditions in Y, and T is the number of records in D.

To better illustrate the calculation of the support measure, let D be a dataset, as shown in Table 2.1, which includes entries about data from pull requests. The rule ***lines_ changed*** = ***"some lines"*** $\wedge$ ***files_ changed*** = ***"many files"*** $\rightarrow$ ***status_ pull*** = ***"accepted"*** has support equal to 50% in D. Considering that $T_{(X \cup Y)} = 4$ (since 4 records satisfy the 3 conditions) and that D has a total of 8 entries ($T = 8$), we can conclude that $Sup = 4/8 = 50\%$.

Table 2.1: Pull request dataset sample.

#	**Project**	**lines_ changed**	**files_ changed**	**status_ pull**
1	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"accepted"*
2	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"accepted"*
3	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"accepted"*
4	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"accepted"*
5	*"ProjectTemplate"*	*"many lines"*	*"many files"*	*"rejected"*
6	*"ProjectTemplate"*	*"many lines"*	*"some files"*	*"rejected"*
7	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"rejected"*
8	*"ProjectTemplate"*	*"some lines"*	*"many files"*	*"rejected"*

On the other hand, *confidence* represents the probability of occurrence of the consequent, given the occurrence of the antecedent. It is obtained in the following manner: $Conf_{(X \rightarrow Y)} = T_{(X \cup Y)}/T_{(X)}$, where $T_{(X)}$ represents the number of records that satisfy the conditions of the antecedent X. Remember that $T_{X \cup Y}$ represents the number of records that the satisfy the conditions in the antecedent and also the conditions in the consequent of the rule.

To understand the calculation of the *confidence* measure, consider the same rule example extracted from Table 2.1. The rule ***lines_ changed*** = ***"some lines"*** $\wedge$ ***files_ changed*** = ***"many files"*** $\rightarrow$ ***status_ pull***=***"accepted"*** has a *confidence* value of 66.6% (4/6) in D, since $T_{X \rightarrow Y} = 4$ (as previously stated) and the conditions which correspond only to the antecedent of the rule ***lines_ changed*** = ***"some lines"*** $\cup$ ***files_ changed*** =

"many files" are satisfied in 6 entries ($T_X = 6$). *Support* and *confidence* are used as a filter in the process of mining association rules, that is, only the rules characterized by having a minimum *support* and *confidence* (defined as an input parameter) are extracted.

Another measure of interest considered in this work is *Lift*, which indicates how more frequently Y occurs given that X occurs. *Lift* is obtained by the quotient of the *confidence* of the rule and the *support* of its consequent, i.e., $Lift_{(X \to Y)} = Conf_{(X \to Y)}/Sup_{(Y)}$. Precisely, the support of the consequent $Sup_{(Y)}$ is the ratio between the number of records that satisfy the conditions in the consequent and the number of records in D. 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 the 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 the occurrence of Y.

Taking into account the rule used to exemplify the *support* and *confidence* measures, $Sup_{(Y)}$ (support of the consequent of the rule) in D is equal to 50% (number of entries that satisfy the condition *status_pull = "merged"*). The *Lift* obtained for this rule is 1.33, since $Lift_{X \to Y} = 66.6/50 = 1.33$, where 66,6% is the *confidence* of the rule. In this case, the result indicates that: when few lines of code are modified through a large number of altered files in a pull request, the chances of having it accepted increases by 33%.

In this work, we use the Apriori [19] algorithm for the extraction of association rules, which is available in the Weka[2] (Waikato Environment for Knowledge Analysis) [32] data mining tool.

2.4 Final Remarks

In this chapter, we presented the main concepts that provide support to the comprehension of this thesis. We discussed the context of pull-based software development and presented each step of this process, from the submission through the determination of the final status of a contribution. As mentioned before, the main previous studies concerning this paradigm are discussed in proper sections in Chapters 3, 4, 5 and 6, as each of them offers a distinct perspective of investigation on pull requests.

We further presented the concepts of association rules and interest measures employed

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

for the evaluation of the relevance, validity, and strength of the mined rules. The understanding of such concepts is crucial to the full comprehension of the methods applied in each of the studies hereby presented.

The following four chapters detail studies about factors that influence: the acceptance/rejection of pull requests (Chapters 3 and 4), their lifetime (Chapter 5), and the assignment of reviewers (Chapters 6). As each of these studies have specific motivations, experimental designs, and relater work, we opted to discuss these aspects in separate chapters.

Chapter 3

Acceptance Factors of Pull Requests in Open-Source Projects

3.1 Introduction

In pull-based software development, we can assume that all involved people have interest in having contributions accepted, if deemed viable for the project. The core team is interested in receiving contributions to alleviate the workload and speed up the release of new features and bug fixes. Similarly, a requester wishes to have his or her contribution accepted for various reasons, such as [33]: altruism, learning experience, or self-use.

What increases the chances of having pull request accepted? Modifying one or several files? Altering few or several lines of code? These questions might be hard to answer if one does not understand the nature of pull requests. Understanding these factors can help developers guide their actions to increase chances of acceptance, as well as help team managers to understand the community around the project.

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 et al. [4], Rahman and Roy [7], and Tsay et al. [6] proposed an exploratory study on the software development model based on pull requests. These works seek to understand the reasons for pull request acceptance and, in some cases, try to foresee the acceptance of a pull request. Their results show rather interesting characteristics: (i) pull requests in projects coded in certain programming languages have a higher rate of acceptance [7]; (ii) a considerable amount of accepted pull requests are analyzed in a short timespan [4]; and (iii) socio-technical factors may also influence on the acceptance of a pull request [6]. However, these works did not explore specifically the characteristics inherent to the pull requests in respect to the projects, to the developer, or even to the contribution itself.

In this study we extract association rules in order to identify new and useful patterns from pull requests data. As a result, we seek to answer the following general research question: Which factors influence the acceptance of pull requests in open-source projects? To do so, we derived the following specific research questions:

RQ 3.1 - Do characteristics of pull requests influence on their acceptance?

RQ 3.2 – Does the combination of pull requests characteristics influence on their acceptance?

RQ 3.3 - Do characteristics of pull requests influence on their fast acceptance?

This study is organized as follows. Section 3.2 discusses the main related works to this exploratory investigation. Section 3.3 presents the research methodology and the data used to perform the experiments. Section 3.4 presents the obtained results. Section 3.5 reports possible threats to the validity of the results. Finally, Section 3.6 concludes the chapter presenting some final remarks and highlighting the main contributions.

3.2 Related Work

Recent studies [4, 7, 6] have been carried out aiming at comprehending the influence of factors over the acceptance of pull requests. In the following, we present each of these studies and highlight the necessity of additional research on the acceptance/rejection of pull requests.

Gousios et al. [4] analyze some factors that contribute to the acceptance of pull request. The results show that decision to accepted a pull request is affected by whether it touches an actively developed part of the system, how large the project's source code base is, and how many files the pull request changes.

Tsay et al. [6] analyze the socio-technical signals associated with accepting pull requests. The results suggest that contributions that include test code are more likely to be accepted. The existence of a social connection between the requester, the project, and the developer responsible for the analysis of the contribution, also influences acceptance 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 they use attributes inherent to the requester, project, and the pull request itself, they do not explore how these factors altogether affect the acceptance of a pull request.

Rahman and Roy [7] performed a comparative study between contributions that were accepted and the ones that were rejected, analyzing a varied collection of data, such as comments, collaboration history, project statistics, and about the developer that filed the pull request. The results of this study show that the programming language in which the pull request was written is relevant to the acceptance. The authors suggest that, in contributions made in Scala, C, C#, and R, a higher acceptance 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 acceptance.

Although important, the approaches presented are limited to analyzing isolated factors, without combining them. In addition, these studies do not suggest, however, the extent of the force that these factors exert in increasing the acceptance rate of pull requests.

3.3 Materials and Methods

In order to answer the research questions raised in Section 3.1, we adopted the association rules extraction, as indicated in Chapter 1. In the following, we detail how we proceeded to answer each research question.

RQ 3.1 - Do characteristics of pull requests influence on their acceptance? In order to answer this question, we mined association rules that contain attributes such as the number of commits, the number of added files, the number of removed files, and the number of modified files in the pull requests. Moreover, we analyzed the correlation between the dominant programming language in the project and the pull request acceptance.

RQ 3.2 – Does the combination of pull requests characteristics influence on their acceptance? For this analysis we conducted the extraction of association rules that combine the influential characteristics of the previous question. In this way, we identified the joint strength of these characteristics on the pull requests acceptance.

RQ 3.3 - Do characteristics of pull requests influence on their fast acceptance? We initially correlated only acceptance and lifetime to verify if the time between the submission and assessment of the pull request influences on acceptance. Next, we analyzed how the pull requests characteristics relate with fast acceptance.

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 dataset was obtained using GHTorrent [18]. The dataset contains 61,592 pull requests from 72 different projects. The projects corpus used in this study is presented in Table 3.1. We collected pull requests sent from November 2010 to December 2013. GHTorrent is able to retrieve data from various dimensions of pull requests stored in GitHub. A large set of attributes are available for gathering (http://ghtorrent.org/files/schema.pdf). In our studies, we conducted experiments with most of these attributes (metadata). However, only a small set of them revealed rules with appropriate semantics in this scenario. The attributes considered in the analysis are shown in Table 3.2.

Table 3.1: Project corpus.

Project	**#PRs**	**Project**	**#PRs**	**Project**	**#PRs**
ActionBarSherlock	238	gitlabhq	1,649	rails	7,820
akka	1,674	hiphop-php	136	redcarpet	120
android	141	html5-boilerplate	526	reddit	415
AutoMapper	78	http-paser	99	redis	453
bitcoin	1,951	impress.js	146	requests	774
boto	1,079	jekyll	653	RestSharp	228
cakephp	1,702	jquery	1,389	revendb	139
chosen	433	knitr	82	sbt	273
CodeIgniter	1,358	Kodi	3,124	scala	3,001
compass	368	libgit2	1,387	scalatra	133
CraftBukkit	1,224	libuv	424	ServiceStack	476
d3	695	MiniProfiler	136	shiny	84
devise	525	mongo	514	Sick_Beard	742
devtools	94	mono	772	SignalR	551
diaspora	1,531	mosh	134	Slim	262
django	1,706	Nancy	794	SparkleShare	157
django-cms	916	netty	600	storm	262
django-debug-toolbar	146	node	2,308	symfony	5,827
elasticsearch	651	octopress	638	ThinmkUp	665
facebook-android-sdk	78	openFrameworks	1,003	three.js	774
finagle	116	paperclip	383	tornado	404
flask	393	phantomjs	406	TrinityCore	1,594
Fonte-Awesome	140	phpunit	313	zf2	4,058
foudation	794	plupload	127	zipkin	238

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

Table 3.2: Attributes used in this study.

Attribute	**Description**
commits_pull	the number of commits in the pull request, with values: *"1 commit"*; *"some commits"*: 2 to 4 commits; and *"many commits"*: more than 4 commits.
developer_type	whether the contributor is internal or external
files_added	the number of added edited files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
files_changed	the number of added, removed, or edited files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
files_edited	the number of edited files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
files_removed	the number of removed files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
first_pull	whether the pull request is the first filed by the requester
language	dominant programming language in the project
life_time	amount of time between the submission and closure of the pull request, where *"very short"* = 1 day; *"short"* = 1 to 3 days; *"medium"* = 3 to 7 days; *"lengthy"* = 7 to 10 days; and *"very lengthy"* = more than 10 days.
status_pull	the final status of the pull request: accepted (merged) or rejected (closed).

In order to transform numerical attributes into discrete ones, we use a unsupervised discretization based on an instance filter available in the Weka tool. Initially, we used the discretization by simple binning, ignoring the class attribute to be defined, but considering frequency distribution. Then, we adjusted the labels for better interpretation of the extracted patterns. All labels have statistical representation in the database.

As stated in Chapter 1, after data selection, we applied data preprocessing, transforming and enriching attributes. Some numeric attributes were discretized so that we could perform the rule extraction. Then, we extracted the rules using the Apriori algorithm available in the Weka tool. In this step, we extracted rules with a minimum *support* of 1% (approximately 600 rules) to ensure that the rules were not obtained by chance, revealing a pattern from the dataset. In addition, we have defined a minimum confidence of 5%. We use these values to get as many rules as possible about the nature of the pull request paradigm. After extracting the rules, we performed the analysis and interpretation of the results. We have conducted this sequence of steps to answer each research questions.

3.4 Results and Discussion

The results presented in this study indicate the power that some attributes have over the pull requests acceptance. In this Section, the main extracted rules are presented and analyzed for each research question.

3.4.1 RQ 3.1 - Do characteristics of pull requests influence on their acceptance?

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

The experiments evidence that some programming languages hold strong influence over the acceptance. Rahman and Roy [7] points out that there is a higher number of acceptance occurrences in some programming languages, but without quantifying the influence that the programming language holds over the acceptance. Through the *Lift* metric of a rule of type *language* $\rightarrow$ *status_pull="accepted"*, which indicates how more frequently the acceptance 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 acceptance of a pull request. Figure 3.1 shows the *Lift* of each rule *language* $\rightarrow$ *status_pull="accepted"*, for each of the programming languages present in the dataset, indicating the increase ($Lift > 1$) or decrease ($Lift < 1$) of the probability of the acceptance.

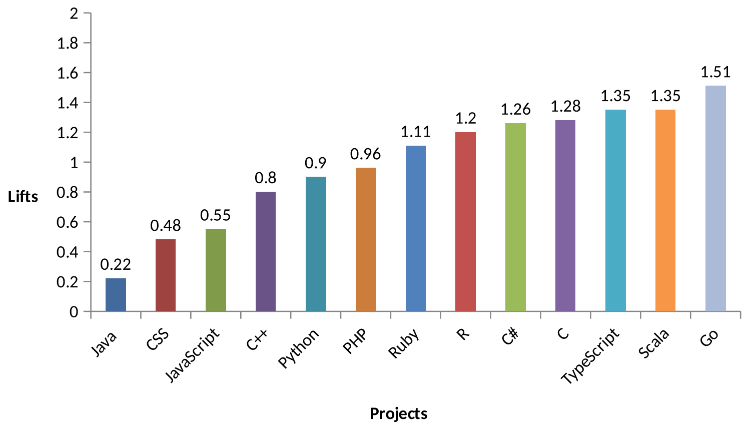


Figure 3.1: *Lifts* of the rules of type: *language* $\rightarrow$ *status_pull* = *"accepted"*.

Figure 3.1 evidences that pull requests written in programming languages like Java, CSS, JavaScript, and C++, have the acceptance chances reduced, whereas C#, C, Type-

Script (few projects), Scala, and Go (few projects) increase the chances of acceptance. The association rules make it feasible to perceive not just the increase/decrease of the chances of acceptance, but also quantifying this variation. By analyzing the *Lift* of the rule *language = "Java → status_pull="accepted"*, it is possible to verify that contributions made in Java have 78% less chance of acceptance (*Lift=0.22*). Similarly, a pull request coded in JavaScript has 45% less probability of acceptance (*Lift=0.55*). Conversely, a contribution written in C# has a chance 26% higher of being incorporated (*Lift=1.26*). We recognize that the requester does not send a contribution according to the language of his preference, but according to the predominant language of the project. Some patterns regarding languages may have been influenced by the number of instances of the database. Our dataset does not present a homogeneous distribution in relation to the language, as shown in Figures 3.2 and 3.3. However, although the patterns are valid for the dataset and indicate an important characterization of the pull request paradigm, additional analysis is required with more projects with higher numbers of pull requests in languages like Go, R, and CSS. These association rules may be useful to help to understand the nature of pull requests giving insight into programming languages in which contributions are more prone to acceptance.

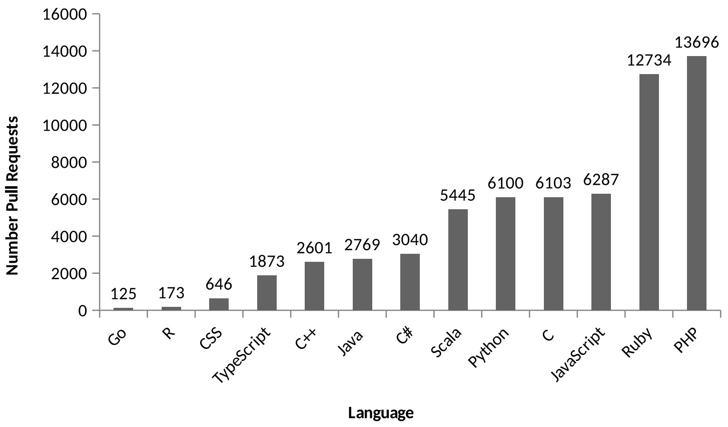


Figure 3.2: Distribution of pull requests by programming language.

languageXProject.eps Another characteristic that affects the acceptance is related to the size of the pull request, represented by the attribute that indicates the number of commits. Figure 3.4 shows the values of the *Lift* metric for rules of type *commits_pull → status_pull = "accepted"* and enables the observation that the higher the number of commits, the smaller the chances of acceptance. This may suggest that the amount of commits heighten the complexity of the analysis.

We also analyzed the values of *Lift* for rules involving the number of files processed

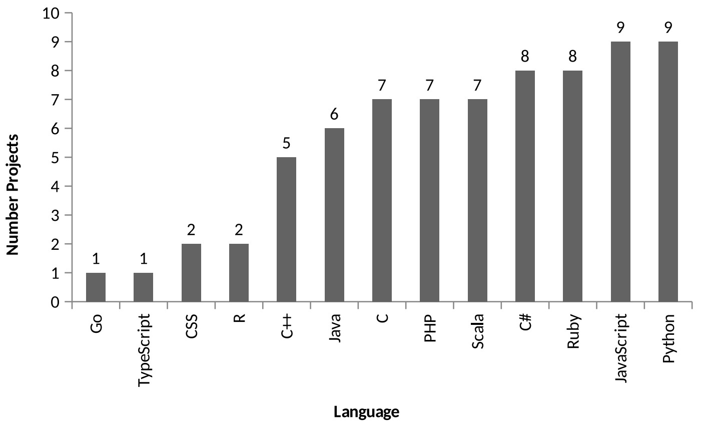


Figure 3.3: Distribution of projects by programming language.

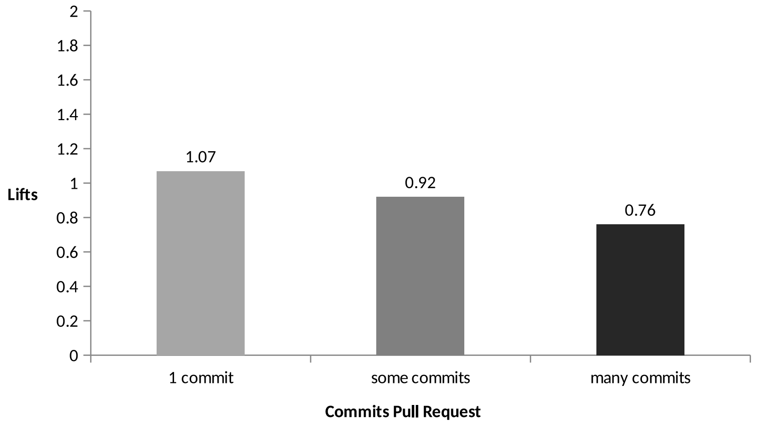


Figure 3.4: *Lifts* of the rules of type: *commits_pull* → *status_pull* = *"accepted"*.

in a pull request in three different dimensions: amount of files added, amount of files removed, and amount of files changed (sum 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 3.5), this attribute does not hold any influence (*Lift*=1.01) over the acceptance, and when there are added files, the chances of acceptance are reduced in 8% (*Lift*=0.92). When there is removal of files (Figure 3.6) in a pull request, we could observe practically no influence, since the values of *Lift* tend to be next to 1. The total number of files changed (Figure 3.7) in a pull request presents, individually, similar behavior as the files removed.

Other rules illustrate the effect of two other attributes over the acceptance: developer type (core team or external collaborator) and whether 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 3.8) have 13% less chance

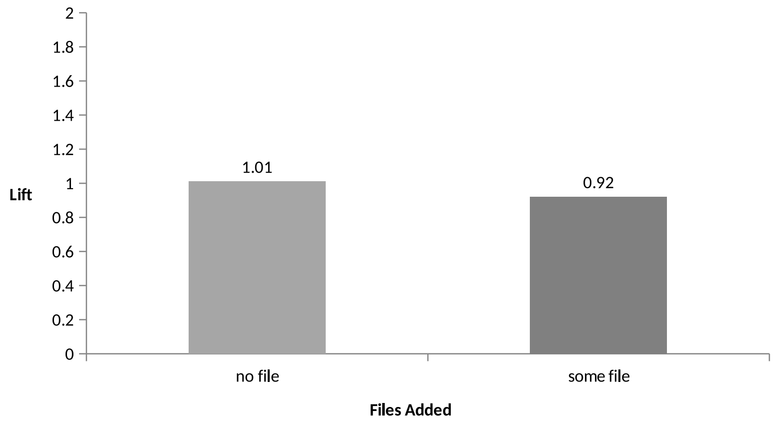


Figure 3.5: *Lifts* of the rules of type: *files_added* → *status_pull* = *"accepted"*.

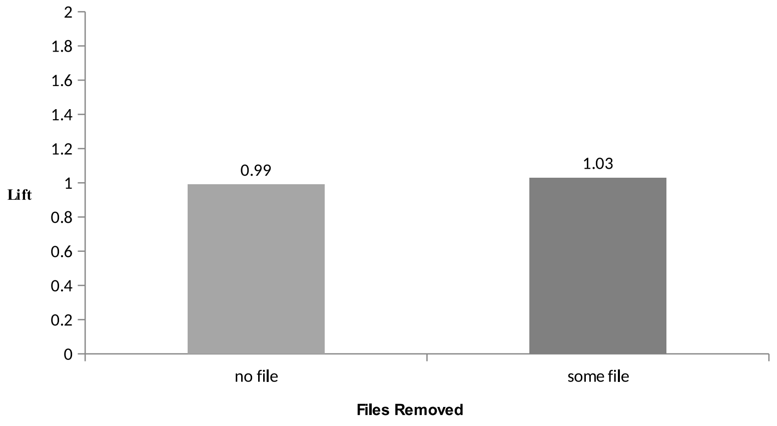


Figure 3.6: *Lifts* of the rules of type: *files_removed* → *status_pull* = *"accepted"*.

of being accepted (*Lift*=0.87), while contributions made by members of the core team increase in 35% the probability of acceptance (*Lift*=1.35).

On the other hand, when the pull request is the first made by a developer (Figure 3.9), the chance of acceptance is decreased in 32% (*Lift*=0.68). 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 acceptance (*Lift*=1.09).

3.4.2 RQ 3.2 – Does the combination of pull requests characteristics influence on their acceptance?

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 acceptance: multiple commits, files added, external developer, and first pull request. This motivates a new research question: Does the

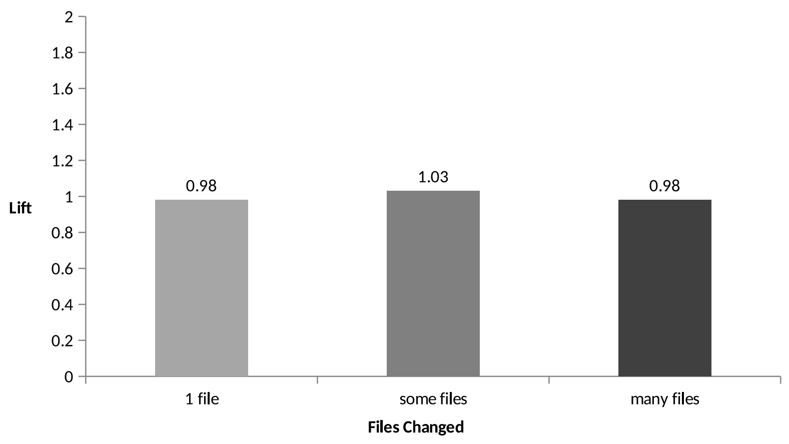


Figure 3.7: *Lifts* of the rules of type: *files_changed* $\rightarrow$ *status_pull* = *"accepted"*.

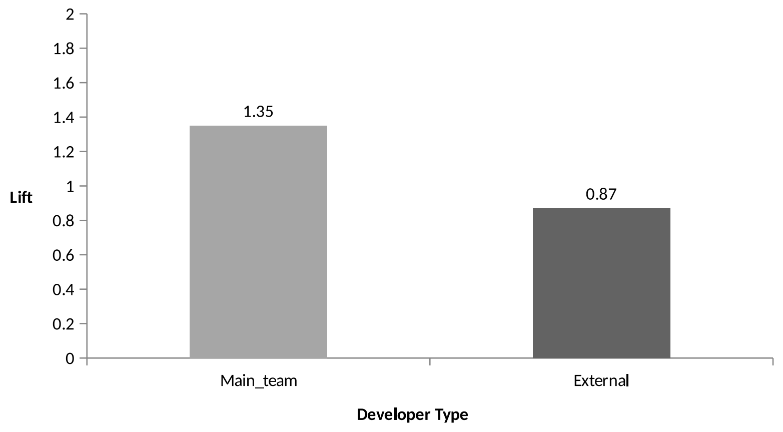


Figure 3.8: *Lifts* of the rules of type: *developer_type* $\rightarrow$ *status_pull* = *"accepted"*.

RQ 3.1 - Do characteristics of pull requests influence on their acceptance?
Answer: Some pull requests characteristics increase the chances of acceptance: language (up to 51%), number of commits (up to 7%), developer type (up to 35%), and requester experience (up to 9%).
Relevance: Our results corroborate with the findings of Gousios et al. [4] and Tsay et al. [6] regarding the number of files. We also confirm the results of Rahman and Roy [7] regarding the correlation between language and acceptance. However, these studies do not suggest the extent of these patterns.
Implications: Based on our results, requesters can adopt practices that reduce the complexity of their contributions (few commits and few files) aiming at increasing the chances of acceptance.

combination of pull requests characteristics influence on their acceptance? Table 3.3 shows some rules (with higher *Lifts* values) with more than one attribute in the antecedent. The consequent of all the presented rules is the condition *status_pull* = *"accepted"* as its value. The rules are sorted according to the order in which the attributes in the antecedent were discussed in Section 3.4.1.

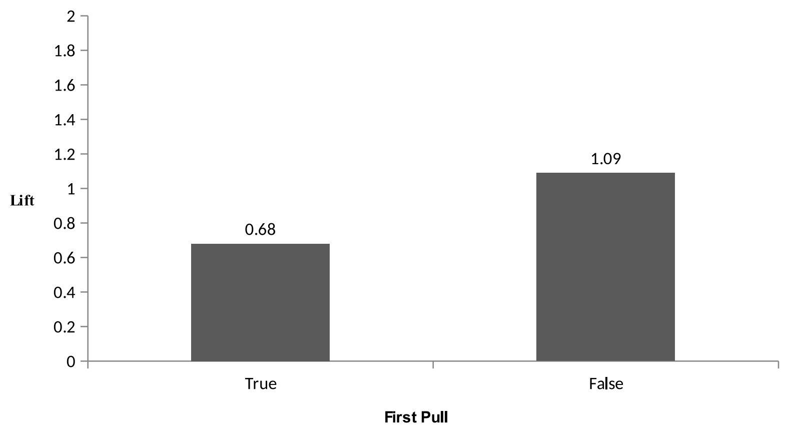


Figure 3.9: *Lifts* of the rules of type: *first_pull* $\rightarrow$ *status_pull* = *"accepted"*.

Table 3.3: Relevant association rules and their measures of interest, where consequent is *status_pull* = *"accepted"*.

#	Antecedent	*Sup* (%)	*Conf* (%)	*Lift*
1	*language* = *"Scala"* $\wedge$ *commits_pull* = *"1 commit"*	4.3	77.0	1.43
2	*language* = *"C#"* $\wedge$ *commits_pull* = *"1 commit"*	2.0	71.0	1.31
3	*commits_pull* = *"1 commit"* $\wedge$ *files_added* = *"no file"* $\wedge$ *developer_type* = *"core_team"* $\wedge$ *first_pull* = *"false"*	10.5	76.1	1.40
4	*project* = *"Akka"* $\wedge$ *commits_pull* = *"1 commit"*	1.5	92.1	1.70
5	*language* = *"Python"* $\wedge$ *developer_type* = *"core_team"* $\wedge$ *first_pull* = *"false"*	1.0	85.0	1.57

The *confidence* of Rule 1 indicates that requests written in Scala that have only one commit are accepted in 77% of the times. The *Lift* of this rule indicates that these two conditions together increase the chances of acceptance in 43%. It is important to observe that considering only the programming language (Scala — see Figure 3.1), the increase in the chances of acceptance was 35%. So the combination of the number of commits and the programming language revealed an important pattern. Rule 2 demonstrates another situation where the number of commits increases the chances of acceptance 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 acceptance increased in

31%, whereas the *Lift* associated with the language C# in relation to the acceptance is only 1.26.

Rule 3 displays a conjunction of factors with positive influence over the acceptance of a pull request. It shows that pull requests with a sole commit not adding new files, made by a core team member who had already made a pull request before, have the chance of acceptance increased by 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 pull requests filed in project Akka, composed by only 1 commit, are accepted in 92% of the cases. In addition, this antecedent increases by 70% the chances of acceptance. 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 acceptance. In this scenario, it is notable that requests made in Python, by members of the core team with some experience with pull requests, have 57% more chance of being accepted.

RQ 3.2 - Does the combination of pull requests characteristics influence on their acceptance?
Answer: The combination of characteristics, which are individually influential, can increase the chances of acceptance (by up to 70%).
Relevance: This was an open question until now, as none of the related work answered it. As far as we know, our study was the first to explore this question.
Implications: Our results showcase situations that increase the chances of acceptance, which should be aimed by requesters. In addition, project managers can enforce these situations.

3.4.3 RQ 3.3 - Do characteristics of pull requests influence on their fast acceptance?

Useful patterns are mined when considering the attribute that indicates how long a pull request takes to be analyzed. Figure 3.10 illustrates the values of *Lift* for the rules that contain the attribute *lifetime* in their antecedent and *status_pull = "accepted"* in their consequent, and displays how more frequently acceptance occurs in relation to the time taken to have the pull request analyzed. It is possible to observe that the increase in the time to analyze a pull request reduces the positive influence on of acceptance of such pull request.

In a complementary analysis, shown in Table 3.4, rules with the attribute *lifetime* in the consequent were mined aiming at identifying attributes that influence on the fast

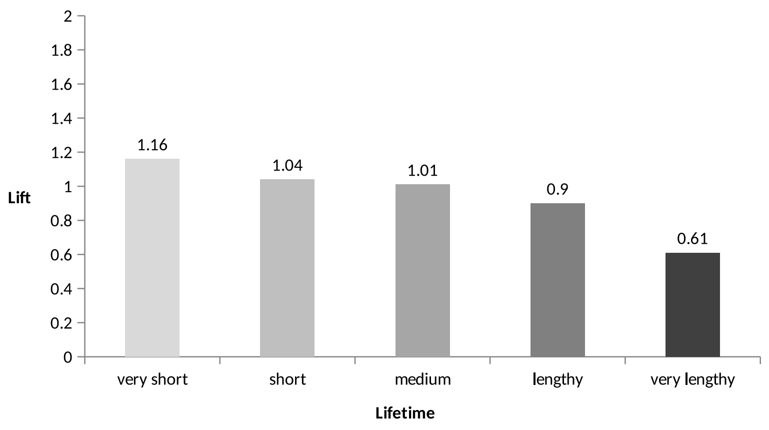


Figure 3.10: *Lifts* of the rules of type: *lifetime* → *status_pull* = *"accepted"*.

acceptance. 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 written in Scala have their chances of fast acceptance (from 1 to 3 days) increased in 216%.

Other results also suggest that some combined factors influence on the very fast acceptance of a pull request. Rule 3, for instance, shows that pull requests with 1 commit, performed by a core team member that have previously submitted pull request are accepted in 24 hours in 49.5% of the times. In addition, this rule indicates that the occurrence of the antecedent increases in 56% the chance of a very fast acceptance.

RQ 3.3 - Do characteristics of pull requests influence on their fast acceptance?
Answer: Results indicate correlation between lifetime and acceptance, where pull requests analyzed within 1 day increase the chances of acceptance in up to 16%. In addition, we have identified some patterns that increase the chances of fast acceptance.
Relevance: Gousios et al. [4] says that the number of commits influence the lifetime of pull requests. However, their work does not suggest the extent of such influence. Tsay et al. [6] and Rahman and Roy [7] do not explore factors involving lifetime and acceptance.
Implications: Our results establish a correlation between two important aspects in the scenario of pull requests, offering insights to investigate factors that influence the lifetime and consequently the acceptance.

3.5 Threats to Validity

Even though we have been careful as to minimize threats to validity of this study, some uncontrolled factors might have influenced the observed results. The experimental study involved the analysis of several projects of varying sizes obtained from an existing dataset

Table 3.4: Association rules with *lifetime* and *status_pull* attributes in the consequent and their measures of interest.

#	Antecedent	Consequent	*Sup* (%)	*Conf* (%)	*Lift*
1	*commits_pull = "1 commit"*	*lifetime = "very short"* ∧ *status_pull = "accepted"*	24.1	37.0	1.17
2	*language = "Scala"* ∧ *commits_pull = "1 commit"*	*lifetime = "short"* ∧ *status_pull = "accepted"*	1.0	17.0	2.16
3	*commits_pull = "1 commit"* ∧ *developer_type = "core_team"* ∧ *first_pull = "false"*	*lifetime = "very short"* ∧ *status_pull = "accepted"*	8.3	49.5	1.56

(MSR Challenge). This might have impacted the generalization made from the results, to some extent, considering that patterns present in projects with a large amount of pull requests – up to thousands – might have superposed patterns that had eventually been present in projects with much less pull requests. We acknowledge the importance of reevaluating the approach with projects with more homogeneous features.

As previously discussed, the projects' dataset was obtained from the MSR challenge. Consequently, we cannot ensure that this dataset precisely reflects the actual projects' data. However, as MSR is a reputable conference, we considered this a minor threat. Nevertheless, we acknowledge the necessity of carrying out our own sampling and processing of attributes originating from software repositories, taking into account the utilization of attributes of different dimensions of data present in the pull requests. In fact, we mitigated this threat during the analyses presented in Chapters 4, 5, and 6 by building our own dataset.

3.6 Final Remarks

The results presented in this study corroborate that association rules mining enables the discovery of useful knowledge to developers and project managers and allow the perception of characteristics that increase the chances of acceptance of a pull request. These rules revealed a behavior that, in some cases, can be followed by a developer that seeks to ensure that the acceptance of his or her pull request occurs. Moreover, this data mining technique makes it possible to discover patterns that provide to the project managers a further comprehension of the features of pull requests submitted to the repository.

We could observe that the following factors have influence in the acceptance of pull requests: programming language, number of commits, external developer, and first pull request. We could also observe that these factors, when combined, may amplify the results. Moreover, we note that these factors also influence on how fast pull requests are accepted. Finally, we could observe that the speed of acceptance is correlated to the acceptance of the pull request.

Chapter 4

Rejection Factors of Pull Requests Filed by Core Team Developers in Software Projects with High Acceptance Rates in Open-Source Projects

4.1 Introduction

In general, developers that are not members of the project but want to contribute somehow file pull requests. However, this mechanism can also aid in systematizing the integration of code produced by core team members, forcing other core team members to review and discuss about the contribution prior to incorporating it into the repository. Pull requests used as a systematization approach for internal contributions have acceptance rates 35% higher [21] – as discussed in Chapter 3. This can be explained by the fact that members of the core team know the project's nature, its issues and maintenance needs, as well as its further evolution expectations. In this context, though less frequently, some contributions are indeed rejected. Within this scenario of internal contribution, important research questions arise:

RQ 4.1 - Do physical characteristics of pull requests influence on their rejection?

RQ 4.2 – Do the requester inexperience and the existence of comments in the pull requests influence on their rejection?

RQ 4.3 - Does the location of artifacts changed by pull requests influence on their rejection?

RQ 4.4 – Does the combination of characteristics influence on the pull request rejection?

The answers to these questions are relevant to the members of the core team. These answers may lead to the improvement of both the way developers contribute and the process of analysis and discussion about the changes made to the software artifacts, with potential positive impact on quality and productivity of the core team.

The goal of this study is to identify, through the extraction of association rules, characteristics that influence on the rejection of pull requests filed by core team members in software projects with high acceptance rates. Complementarily to the extraction of rules, we also did a qualitative analysis in pull requests of some projects, aiming at finding a justification for the discovered patterns.

The results of our research indicate that some key factors increase the chances of having internal contributions rejected: (i) physical characteristics and the complexity of contributions, as well as the location of the artifacts that have been modified; (ii) previous experience with pull requests; and (iii) contribution policy of the project.

This study is organized as follows. Section 4.2 presents the main works related to this research. In Section 4.3, we present the method used for carrying out the research and the dataset used in the experiments. Next, in Section 4.4, we report the obtained results and discuss their implications. Section 4.5 presents the threats to validity of the results. Finally, Section 4.6 concludes this work by highlighting its main contributions and presenting some future work.

4.2 Related Work

Recently, some researches have studied GitHub to identify factors that influence on the acceptance/rejection of pull requests [4, 6, 7].

Gousios et al. [4] investigated the popularity of the pull-based development paradigm, its life cycle, the factors that contribute to the integration of a pull request, and the time necessary to do it. The work reveals that there is no clearly outstanding reason for the rejection of pull requests. However, they group reasons that have a timing dimension (obsolete – 4%, conflict – 5%, superseded – 18%), where 27% of pull requests are rejected due to concurrent modifications of the code in project branches. Another 16% (superfluous – 6%, duplicate – 2%, deferred – 8%) is rejected as a result of the contributor not having identified the purpose of the project correctly.

Tsay et al. [6] suggest that the contributions that include test code are more likely to be incorporated. Moreover, they indicate that factors such as the number of modified files and the amount of comments throughout the discussion reduce the chances of incorporation of such a contribution.

Rahman and Roy [7] did a comparative study between pull requests that were accepted and the ones that were rejected. The results show that the programming language in which the pull request was written has significant influence on acceptance. The authors suggest that pull requests written in Scala, C, C#, and R have a higher acceptance rate, while the ones written in other languages are more prone to rejection. The age and maturity (i.e., number of forks) of the project affect the rejection of pull requests. They also indicate that with the increase in the number of forked projects, the average number of pull requests per month increases. Moreover, with the increase in the number of forked projects, the rejection rate of pull requests increases especially for projects with more than 2,000 forks. The number of developers involved in a project and their experience can affect the rejection rates of pull requests for a project. This study stated that the average number of pull requests per month for a project increases almost regularly with the increase of the number of developers. However, the rate of unsuccessful pull requests increased exponentially for a project with more than 4,000 developers involved.

In a previous work [21], we have identified that physical characteristics have influence on the acceptance/rejection of pull requests. The results indicate that the higher the number of files that had been modified and the higher the number of commits, the lower the chances of having the pull request accepted. For example, pull requests with many commits have 24% less chances of acceptance. Moreover, pull requests written in Java have 78% less chance of acceptance. Similarly, a pull request coded in JavaScript has 45% less probability of acceptance.

Silva et al. [34] investigated technical reasons for rejection of pull requests. They identified seven technical debt categories, namely: design, documentation, test, build, project convention, performance, and security debt. The results indicate that the most common category of technical debt in pull requests rejected is design with 39.34%, followed by test with 23.70% and project convention with 15.64%. The results also reveals that the type of technical debt influences on the size of the pull request discussions, e.g., security and project convention debts instigate more discussion than the other types.

None of the works described in this Section specifically analyzed the forces that lead to the rejection of pull requests filed by core team members, especially in projects with elevated acceptance rates.

4.3 Materials and Methods

The experimental process of our study can be divided into three main phases. In the first phase we extracted general association rules – as described in the Chapter 1 – from 20,140 pull requests from 7 projects that use this paradigm to systematize internal contributions and present high acceptance rate. We extracted rules with a minimum *support* of 0.2% (approximately 40 rules) to ensure that the rules were not obtained by chance, revealing a pattern from the dataset. In addition, we have defined a minimum confidence of 5%. We use these values to get as many rules as possible about the nature of the pull request paradigm. We collected pull requests filed from November 2010 to July 2015. In this phase, we consider pull requests from all the collected repositories and answered RQ 4.1 and RQ 4.2. In the second phase, we evaluated two projects individually in order to answer RQ 4.3. Next, in order to answer RQ 4.4, we evaluated the joint influence of the number of commits, number of files, and requester inexperience on rejection in complete corpus and we also used the projects Akka and IPython to qualitatively investigate some patterns. The paragraphs in the following describe the analyses that we made to answer each of the research questions.

RQ 4.1 - Do physical characteristics of pull requests influence on their rejection? In order to answer this question, we mined association rules that contain attributes about physical characteristics, such as the number of commits and the number of modified files in the pull requests.

RQ 4.2 - Do the requester inexperience and the existence of comments in the pull requests influence on their rejection? We extracted rules in order to detect relationships between requester inexperience with pull requests rejection. We also analyzed the influence of the comments in pull requests on their rejection.

RQ 4.3 - Does the location of artifacts changed by pull requests influence on their rejection? We extracted rules from two individual projects: Akka and IPython. This analysis was performed on specific projects because location attributes (files and directories) are specific to each of them. In the location analysis, since the amount of changed files and directories is large, we employed a technique for attribute selection (Pearson Correlation) over them, choosing the 10 more selective files and directories in relation to the pull request final status.

RQ 4.4 – Does the combination of characteristics influence on the pull request rejection? For this analysis we conducted the extraction of association rules in which the influential characteristics of the previous questions were combined. In this way, we identified the joint strength of these characteristics on the pull requests rejection.

From these 20,140 pull requests, 93.42% were accepted and only 6.58% were rejected. All pull requests filed by external contributors (559 pulls requests) were discarded, as this is not the focus of our study. Table 4.1 shows some characteristics of the analyzed projects: number of pull requests, acceptance rate, and rejection rate. As mentioned before, we conducted experiments with most of the attributes extracted via GHTorrent. However, only a small set of them revealed rules with appropriate semantics in this scenario. The attributes considered in the analysis are shown in Table 4.2.

Table 4.1: Characteristics of projects analized.

Project	**Language**	**#Pull Requests**	**Accepted**(%)	**Rejected**(%)
Akka	Scala	954	86.2	13.8
Bitcoin	C++	1,563	83.4	16.6
Brackets	Java Script	3,685	93.0	7.0
Commcare-hq	Python	6,961	98.9	1.1
IPython	Python	3,163	84.7	10.3
Katello	Ruby	1,046	94.3	5.7
Kuma	HTML	2,768	92.3	7.7

In order to transform numerical attributes into discrete ones, we use the discretization by simple binning, ignoring the class attribute to be defined, but considering frequency distribution. All labels have statistical representation in the database.

4.4 Results and Discussion

From the extracted rules, we could identify patterns that increase the chances of a pull request filed by core team members being rejected. These patterns tend to have an even more powerful indication of rejection if they are related to the physical characteristics, project's collaboration policy, and with the location of the changed artifacts. In this section, we present and discuss the major results obtained from this study.

Table 4.2: Attributes used in this study.

Attribute	**Description**
comments_pull	whether the pull request contain or not comments.
commits_pull	the number of commits in the pull request, with values: *"1 commit"*; *"some commits"*: 2 to 4 commits; and *"many commits"*: more than 4 commits.
directory_names	represents a set of 10 attributes, where each is the name of a directory. The value "true" indicates that a file from that directory was changed, the value "false" indicates otherwise.
documentation	indicates if pull request has associated documentation.
file_names	represents a set of 10 attributes, where each is the name of a file. The value *"true"* indicates that a file was changed, the value *"false"* indicates otherwise.
files_changed	the number of added, removed, or edited files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
first_pull	whether the pull request is the first filed by the requester.
status_pull	the final status of the pull request: *"accepted"* (merged) or *"rejected"* (closed).
test	indicates if pull request has associated test.

4.4.1 RQ 4.1 - Do physical characteristics of pull requests influence on their rejection?

The analysis of association rules suggests that some individual pull request characteristics influence on the rejection of internal contributions.

A factor that influences on the rejection of pull requests filed by core team members is the number of commits present in the pull request. Figure 4.1 shows the *Lift* values for association rules of type *commits_pull* → *status_pull* = *"rejected"*. Pull request with only one commit the chances of rejection decrease in 14% (*Lift*=0.86). In pull requests with several commits (*commits_pull* = *"many commits"*), the chances of rejection increase in 51% (*Lift*=1.51). Thus, we can conclude that the greater the number of commits performed in a pull request, the higher is the chance of rejection.

Similarly to the number of commits, the increase in the number of changed files (added, edited, or deleted) in the pull request also increases the chances of rejection. Figure 4.2 displays the *Lifts* of rules of type *files_changed* → *status_pull* = *"rejected"*. Pull request that changed only one file the chances of rejection decrease in 14%(*Lift*=0.96) When many files are modified in a pull request, it is 44% more likely to be rejected (*Lift*=1.44).

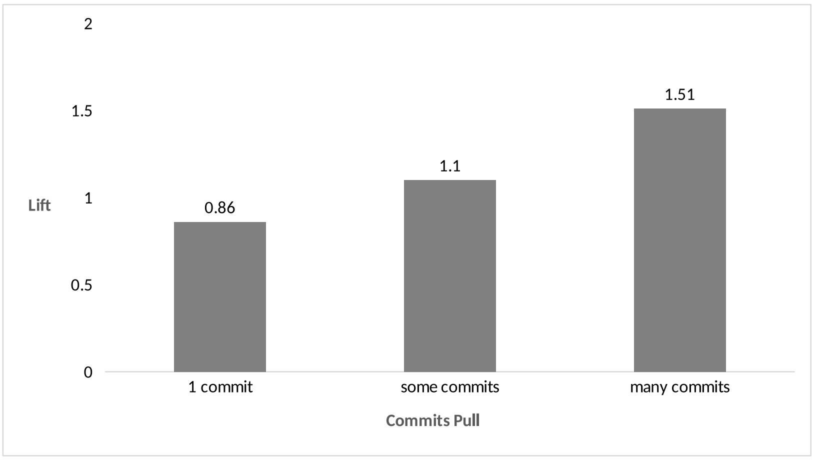


Figure 4.1: *Lifts* of the rules of type: $commits\_pull \rightarrow status\_pull$ = *"rejected"*.

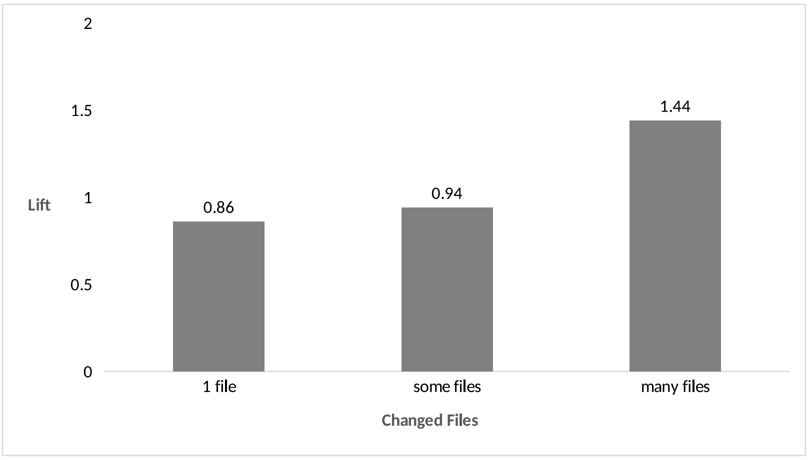


Figure 4.2: *Lifts* of the rules of type: $files\_changed \rightarrow status\_pull$ = *"rejected"*.

4.4.2 RQ 4.2 - Do the requester inexperience and the existence of comments in the pull requests influence on their rejection?

Submitting the first pull request is another factor that has influence on the rejection of the contribution. Figure 4.3 shows the *Lifts* of the rules in which $first\_pull$ is the antecedent and $status\_pull$ = *"rejected"* is the consequent. The analysis of the *Lifts* reveals that, when the first contribution of a developer occurs via pull request ($first\_pull$ = *"true"*), the chances of rejection are 3.38 times higher. Notwithstanding, when this is not the case ($first\_pull$ = *"false"*), the pull request has 21% less chances of being rejected. This way, we can conclude that using the pull request paradigm for the first time increases the chances of rejection.

The existence of discussion or not during the review of the pull request is another factor that impacts the rejection rate. When there is a discussion about the pull request ($comments\_pull$ = *"has comments"*), the chances of rejection are 13% higher. When this

RQ4.1 - Do physical characteristics of pull requests influence on their rejection?
Answer - There exists a correlation between rejection of pull requests and their physical characteristics. A large number of commits in a pull request raises the chances of rejection by 51%, while a large number of files raises the chances by 44%.
Relevance: Our results indicate that the influence of the number of commits and number of files on pull requests rejection, already found by Soares et al. [21] and Tsay et al. [6] in a more generic population, also holds for internal contributions. In addition, by*Lift* our results quantify the extent of such influence. Gousios et al [4] did not explore this question.
Implications: Our results indicate that complex contributions tend to rejection. This knowledge reveals to the core team members a pattern to be avoided, and consequently, improve internal contribution activities.

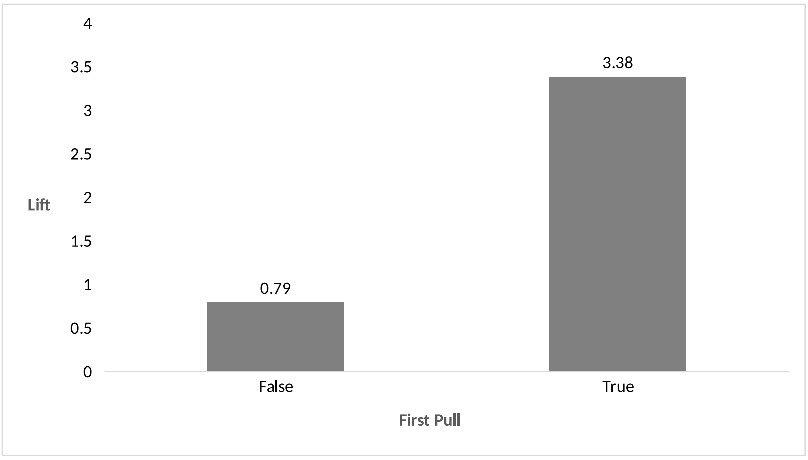


Figure 4.3: *Lifts* of the rules of type: *first_pull* $\rightarrow$ *status_pull* = *"rejected"*.

does not occur, the chances of rejection are reduced in 6%. Figure 4.4 shows this pattern and illustrates the *Lift* values for rules of type *comments_pull* $\rightarrow$ *status_pull* = *"rejected"*.

4.4.3 RQ 4.3 - Does the location of pull requests influence on their rejection?

As described in Section 4.3, we extracted association rules from two software repositories, separately in order to investigate if the location of the pull request also affects its rejection. As discussed before, this analysis needed to be performed in a project basis because location attributes (files and directories) are specific to each project. Next, we present the results obtained with the investigation carried out in the repositories of projects Akka and IPython.

In this analysis, we considered the location (files and directories) of the changes. The results show that these factors influence on the rejection of internal contributions. In

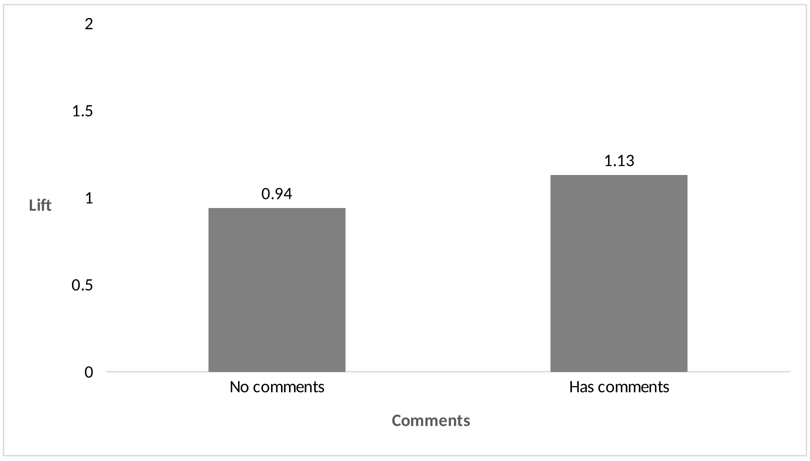


Figure 4.4: *Lifts* of the rules of type: *comments_pull* → *status_pull* = *"rejected"*.

RQ4.2 - Do the requester inexperience and the existence of comments in the pull requests influence on their rejection?
Answer - The inexperience of the core team member who submits pull requests increases more than twice the chances of rejection. In addition, pull requests with comments during their review also increase up to 13% their chances of rejection.
Relevance: Our results confirm that the influence of comments and the requester inexperience on pull request rejection, respectively discussed by Tsay et al. [6] and Rahman and Roy [7] in a more generic population, also holds for internal contributions. In addition, we quantify the extent of influence through *Lift* measure. Gousios et al. [4] did not explore this question.
Implications: Although core team members have push privileges to the repository, they use the pull request paradigm as a quality assurance strategy, forcing another team member to review their changes. The inexperience of the team members may indicate that they have not yet adequately understood the operation of the pull requests model. Project managers could promote training or produce materials with examples indicating how a pull request should be filed.

addition, the location proves to influence the rejection of pull requests in repositories that adopt a structure of organization based on directories. From this perspective, additionally, we qualitatively observed some of the patterns found, in order to which presents justification for their existence from the analysis of the collaboration policies employed by the projects.

By extracting rules containing location attributes from Akka project, we noticed that, depending on where the modifications occurred, the chance of rejection can be higher. Table 4.3 shows rules that unravel knowledge on rejection factors in the contributions to project Akka. Rule 1, for instance, suggests that when modified artifacts are located in the directory *"…/scala/akka/remote"*, the pull request has 173% (*Lift* = 2.73) more chances of being rejected. Rule 2 in Table 4.3 indicates that when the file *"…/camel/camel.scala"* is present in the contribution, the probability of rejection increases in 261% (*Lift* = 3.61).

Table 4.3: Relevant association rules for rejection (consequent *status_pull = "rejected"*) from project Akka.

#	Antecedent	*Sup*	*Conf*	*Lift*
1	*directory_names = ".../scala/akka/remote"*	1.0	38.0%	2.73
2	*file_names = ".../camel/camel.scala"*	1.0	50.1%	3.61

The rules inferred from the IPython repository also reveal that location is a factor of influence on the rejection of contributions. For example, Rules 1 through 3 on Table 4.4 show the values of the interest measures for specific antecedents. Rule 1 indicates that pull requests that modify artifacts contained in the main directory have an increase of 54% in the chances of being rejected.

Table 4.4: Relevant association rules for rejection (consequent *status_pull = "rejected"*) from project IPython.

#	Antecedent	*Sup*	*Conf*	*Lift*
1	*directory_names = "IPython/core"*	3.0	16.0	1.54
2	*directory_names = "IPython/extensions"*	1.0	20.0	1.99
3	*file_names = "IPython/core/magic.py"*	1.0	26.0	2.55

From the individual analysis of the projects that make use of pull requests for the systematization of internal collaboration, it was possible to perceive that, in some cases, the location of the contributions have strong influence on the chances of rejection.

RQ4.3 - Does the location of artifacts changed by pull requests influence on their rejection?
Answer - The location of artifacts modified in pull requests influence in their rejection. In some projects, the location may increase the chances of rejection in up to 261%.
Relevance: Previous works did not investigate this questions. As far as we are aware, our study was the first to bring forth results that indicate the influence the pull request location on the rejection when it is filed by core team members in software projects with high acceptance rates.
Implications: Our results indicate that some codebase regions are sensitive to changes and tend to rejection of pull requests. This knowledge is important for project managers and developers to be aware of the need to share information about critical areas of the software.

4.4.4 RQ 4.4 – Does the combination of characteristics influence on the pull request rejection?

The results presented so far are relative to rules with only one attribute in the antecedent. These attributes, on their own have an influence on the pull requests rejection. This motivated an analysis in order to verify if there is a joint influence of these factors on the rejection.

We were able to identify compound rules, relating together pull request physical characteristics and requester inexperience (see Table 4.5). Despite being less frequent in the dataset – which can be justified by the high acceptance rate of the projects that have been investigated – the rules suggest that the conjunction of influencing factors on the rejection of pull requests increases even more the chances of rejection, as one would expect.

Table 4.5: Relevant association rules for rejection (consequent *status_pull = "rejected"*) and their measures of interest.

#	Antecedent	*Sup* (%)	*Conf* (%)	*Lift*
1	*first_pull = "true"* ∧ *files_changed = "many files"*	0.44	37.0	5.66
2	*first_pull = "true"* ∧ *commits_pull = "many commits"*	0.40	39.1	5.92
3	*first_pull = "true"* ∧ *commits_pull = "many commits"* ∧ *files_changed = "many files"*	0.24	45.1	6.84

Rule 1, for instance, indicates that when a developer submits a pull request for the first time and modifies many files, his or her contribution will be rejected in 37% of the time. Furthermore, this rule's *Lift* value shows that the occurrence of the antecedent increases in more than 5 times (*Lift=5.66*) the chances of having the pull request rejected. Rule 2 evidences that when a core team member submits a pull request for the first time with many commits, the chances of rejection increases in almost 6 times (*Lift=5.92*) the usual rejection rate of these projects. Rule 3 features another influencing scenario in the rejection of pull requests. One can observe that the requester's lack of experience, associated with the volume of commits and modified files, increases by almost 7 times (*Lift=6.84*) the chances of rejection. This complements the knowledge about the relation between the sizes of modifications, thus revealing a pattern that must be avoided in projects that employ the pull request approach to systematize internal contributions.

Table 4.6 shows the *Lift* values for the previously discussed rules the Sections 4.4.1 and 4.4.2 considering the projetcs Akka and IPython, and the complete corpus. In general, all the patterns continue to impact the rejection of pull requests in the projects analyzed here. Notwithstanding, what varies in this analysis is the intensity of the influence. The rules are sorted according to the order of presentation in the previous Sections 4.4.1 and 4.4.2. Results allow the conclusion that the factors identified as influential in the previous analysis are also valid when the pull requests of each project are considered separately.

Table 4.6: Relevant association rules for rejection (consequent *status_pull* = *"rejected"*) in projects Akka, IPython, and in the complete corpus.

#	Antecedent	*Lifts*		
		Akka	IPython	All
1	*commits_pull* = *"many commits"*	2.06	1.27	1.51
2	*files_changed* = *"many files"*	1.23	1.32	1.44
3	*first_pull* = *"true"*	1.72	2.49	3.78
4	*first_pull* = *"true"* ∧ *commits_pull* = *"many commits"*	3.92	3.57	5.92
5	*first_pull* = *"true"* ∧ *files_changed* = *"many files"*	2.74	4.51	5.66
6	*first_pull* = *"true"* ∧ *commits_pull* = *"many commits"* ∧ *files_changed* = *"many files"*	4.70	5.14	3.81

We also extracted rules to verify whether rejection of pull requests can be influenced by the association between physical factors, requester inexperience, and location. In project Akka, it is also noticeable that the conjunction of factors has impact on the rejection of pull requests. When observing Rules 1 and 2 in Table 4.7, we note that the location and the size of the contribution favor, in some cases, the rejection. For example, Rule 2 evidences that when *first_pull* = *"true"* and *commits_pull* = *"many commits "* and *file_names* = *".../camel/camel.scala"* occur in a pull request, the chances of rejection are raised by 623% (*Lift* = 7.23). This information reveals undesirable features in pull requests filed by core team members.

Table 4.7: Relevant association rules for rejection (consequent *status_pull* = *"rejected"*) in project Akka.

#	Antecedent	*Sup*	*Conf*	*Lift*
1	*first_pull* = *"true"* ∧ *directory_names* = *".../scala/akka/remote"*	1.1	75.2	5.42
2	*first_pull* = *"true"* ∧ *commits_pull* = *"many commits"* ∧ *file_names* = *".../camel/camel.scala"*	1.0	100.0	7.23
3	*first_pull* = *"true"* ∧ *files_changed* = *"many files"* ∧ *documentation* = *"false"*	2.1	40.0	2.86

Through tapping into location attributes, we were able to know what type of file is modified. Project Akka uses, by default, an RST (ReStructuredText) file markup syntax for the documentation of contributions. Consequently, pull requests that alter (insert or edit) files with a given extension have specific documentation about them. Furthermore, the chances of rejection of pull requests on project Akka when they do not have proper documentation are 10% higher (*Lift* = 1.10). Rule 3 of Table 4.7 points out that if the first contribution of a given developer modifies various files and does not possess documentation, the chances of rejections increase in 186% (*Lift* = 2.86). A qualitative analysis of the repository can explain this pattern. The collaboration policy of this repository is clear about the preference for documented pull requests: *"All documentation is preferred to be in Typesafe's standard documentation format ReStructuredText, and compiled using Typesafe's customized Sphinx based documentation generation system"*[1].

The conjunction of factors that have influence on the rejection of pull requests also proves to be important in the IPython project. Rule 1 of Table 4.8, for example, implies that pull requests that altered artifacts in the *IPython/core* directory and did not have any comments during the review step of the contribution are subject to rejection. In this case, the chances of the pull request being rejected increase in 66% (*Lift* = 1.66). Although the existence of comments usually increases the chances of rejection, it is not valid for the IPython project. The occurrence of this pattern can be explained by a qualitative analysis on the repository. The recommendations made about the collaboration process suggest that a large number of comments during the discussion is desirable: "*Review and discussion can begin well before the work is complete, and the more discussion the better*"[2].

Table 4.8: Relevant association rules for rejection (consequent *status_pull = "rejected"*) in project IPython.

#	Antecedent	*Sup*	*Conf*	*Lift*
1	*directory_names = " IPython/core"* ∧ *comments_pull = "no comments"*	2.1	17.3	1.66
2	*first_pull = "true"* ∧ *files_changed = "many files"* ∧ *test = "false"*	1.0	63.2	6.08

When analyzing the location of artifacts that have been modified in the IPython repository, the directory *test* appears quite frequently, indicating that files present in it represent test cases. The submission of pull requests that contain test cases is stimulated by the collaboration policy of the repository, which suggests that: *"pull requests should be tested"*[3]. This acknowledgment justifies the existence of the pattern evidenced by Rule 2 in Table 4.8. In this case, this pattern is strengthened by the conjunction of influencing factors towards the rejection. Thus, in the IPython repository, when a core team member

[1]https://github.com/akka/akka/blob/master/CONTRIBUTING.md
[2]https://github.com/IPython/IPython/wiki/Dev:-GitHub-workflow
[3]https://github.com/ipython/ipython/blob/master/CONTRIBUTING.md

files a pull request for the first time, modifies many files, and does not add test cases, the possibility of rejection increases in 500% (*Lift* = 6.08).

RQ 4.4 - Do the combination of characteristics influence on the pull request rejection?
Answer: The combination of characteristics that are individually influential can increases the chances of rejection (in up to 584%). This influence can be even greater when it involves the pull request location (in up to 623%). Moreover, we found that the contribution policy of the some projects justifies some influential patterns of rejection.
Relevance: This was an open question until now, as none of the related work provided answers to it. As far as we know, our study was the first to explore this question in the scenario where pull requests are filed by core team members.
Implications: The results suggest that core team members should double care when submitting pull requests that touch files in specific directories. Moreover, they should avoid filing their first pull request with a large number of files and commits.

4.5 Threats to Validity

Although we have taken care to reduce the threats to validity in our study as much as we could, a few uncontrolled factors may have influenced the observed results.

Regarding the correctness of findings, our small corpus (seven projects) led to some association rules having small *support*. Therefore, some of them may have been found accidentally. On the other hand, we have identified a set of patterns whose frequent occurrence is difficult to accredit to chance. Yet, we recognize the need to confirm all of our findings with a broader study involving a multitude of projects.

We also acknowledge that the ability to generalize our findings is restricted to projects containing the common characteristics of our selected projects: all of them are open source, have high acceptance rate, and the majority of pull requests were filed by core team developers. Thus, we cannot generalize our results to industrial projects or open source projects with different characteristics. Our analysis, though, has revealed patterns that may be present in other projects. Additional studies are needed to assess this.

4.6 Final Remarks

In this chapter we presented a study about the factors that increase the chances of having pull requests filed by core team members rejected, considering projects with high acceptance rates. The analyses unravel patterns that should be avoided by core team members to reduce the chances of rejection of their contributions. Particularly, the results allow the following conclusions: (i) the requester's experience, location, and the number of commits strongly influence the rejection of internal contributions; (ii) the combination of the aforementioned characteristics increases the chances of rejection; and (iii) specific project recommendations, such as mandatory discussion and inclusion of tests in pull requests, may lead to the rejection of the contribution when ignored.

Chapter 5

Influential Factors on the Lifetime of Pull Requests in Open-Source Projects

5.1 Introduction

Understanding the factors that influence on lifetime of pull requests is important to both the contributors and the reviewers. As previously discussed, the acceptance of a pull request is the main goal of a contributor and is also a usually desired effect for the reviewers. The former invests time and effort, and expects to see the contributions incorporated into the product as soon as possible, while the latter benefits from the contributions, as they improve the software being developed and alleviate the pending tasks. However, the lifetime of a pull request, comprising the time interval between its opening and its closing, via acceptance or rejection, has a high variance [4, 9]. Extended delays for pull requests that are probably going to be accepted are undesired, as they may affect the contributor engagement in the collaboration process [35] and postpone the introduction of important bug fixes and features to the product.

Existing works [4, 9] attempted to identify aspects that could be related to the lifetime of pull requests. These works considered only few attributes of pull requests in their analysis of lifetime, such as: static characteristics about pull requests and few features about the external contributors, the number of modified files, the number of comments, and the external contributors acceptance rate. They left aside other important attributes, such as: locations of altered files, the developer responsible for the review process, and the social relationships among external contributors and reviewers.

The goal of our study is to identify patterns that indicate influential factors on the lifetime of pull requests regarding the socio-technical aspects of the pull request, the external contributors, and the contribution process. More specifically, we aim at answering the following research questions:

RQ 5.1 - Is there an effective relationship between the lifetime of pull requests and their acceptance?

RQ 5.2 - Do physical characteristics of pull requests influence on their lifetime?

RQ 5.3 - Does the location of pull requests influence on their lifetime?

RQ 5.4 - Does the profile of external contributors influence on the lifetime of pull requests?

RQ 5.5 - Do characteristics related to the review process influence on the lifetime of pull requests?

We first collected data from 30 projects and, with the aid of association rules, we identified some features that, in an isolated or combined way, help to explain the lifetime of pull requests. Moreover, we also analyzed qualitative aspects to amplify the comprehension of some patterns. For instance, we could observe that: (i) contributions with shorter lifetimes tend to be accepted into the main repository, while the slower ones tend to be rejected; (ii) physical characteristics such as number of commits, changed files, and lines of code have influence, in an isolated or combined way, on the pull request lifetime; (iii) the changed files and the directories where they are stored can be robust predictors for pull request lifetime; (iv) the profile of external contributors and his or her social relationships have influence on lifetime; (v) the amount of comments in a pull request, as well as the developer responsible for the review process, are important predictors for lifetime duration.

The remainder of this Chapter is organized as follows. Section 5.2 displays the main works related to the study of pull request lifetime. In Section 5.3, we present the research process and the dataset used in the experiments. In Section 5.4, we show the obtained results and discuss their implications, answering each of the aforementioned research questions. In Section 5.5, we present the threats to validity of this study. Finally, Section 5.6 concludes this study highlighting our main contributions.

5.2 Related Work

Recently, some studies [4, 9, 21] proposed to investigate the lifetime of pull requests. In the following paragraphs, we discuss these studies.

Gousios et al. [4] investigated the popularity of the pull request paradigm, as well as its lifecycle, the characteristics that influence the integration of a contribution, and the time needed to do so. This quantitative study identified that the time distribution for merges (accepting the pull request) is highly unequal. It also indicated that 66.83% of all accepted pull requests have lifetimes of up to 3 days long and accepted pull requests have significantly shorter lifetimes in comparison with the rejected ones. This study reveals that pull requests submitted by core team members present shorter lifetimes in comparison with the ones from external contributors. They also conclude that time is strongly related to the external contributors history of acceptance. This means that the more a developer has pull requests accepted, the shorter the lifetimes of future pull requests. In spite of being a pioneer work on the lifetime of pull requests, their study does not consider the combination between various dimensions of the data present in this scenario (regarding pull request, requesters, reviewers and discussion). Furthermore, they did not measure the degree of influence that the characteristics identified as relevant have over lifetime.

Yu et al. [9] investigated what they call "determinants of pull request evaluation latency" through a qualitative study. This study evidences that the amount of comments is the main predictor for lifetime. This study also concludes that physical and social features may have influence on lifetime. Additionally, it suggests that simpler contributions possess less latency in the review process. This work, however, does not measure the degree of influence of such characteristics and does not evaluate the joint contribution of these characteristics on the pull request lifetime. Moreover, they do not consider important physical characteristics such as location (i.e., which directories and files were changed).

When investigating acceptance factors of pull requests, we [21] preliminarily identified that lifetime can, to some extent, influence the acceptance of pull requests. We discovered that pull requests evaluated within 24 hours has 16% more chances of acceptance, while pull requests with lifetime between 7 and 10 days have 10% less chances of acceptance. Similarly, pull requests lasting more than 10 days have 39% less chances of acceptance. It is possible to observe that the increase in the time needed to analyze a pull request reduces the chances of acceptance. This motivates us to investigate the root causes of the pull requests lifetime.

Although there are already important assessments of lifetime in the literature, the existing results do not take into consideration other several characteristics intimately related to pull requests, such as the location, the reviewer, and the combination of different dimensions of the existing data such as commits and files, and commits and comments. Besides, there is little quantitative evidence effectively measuring the extent of the influence that these characteristics have on lifetime.

5.3 Materials and Methods

In order to answer the research questions raised in Section 5.1, we also adopted the association rules extraction, as indicated in Chapter 1. Moreover, considering the multidimensionality of pull request data, we performed additional analyses to support the obtained answers. We briefly discuss in the following paragraphs which analyses we did to answer each of the research questions.

RQ 5.1 - Is there an effective relationship between the lifetime of pull requests and their acceptance? In order to answer this question, we extracted association rules involving these two attributes, where *lifetime* is the antecedent and *status_pull* is the consequent. The results of this analysis are presented in Section 5.4.1.

RQ 5.2 - Do physical characteristics of pull requests influence on their lifetime? For investigating physical characteristics of pull requests, we needed to conduct an extraction of rules that contained attributes counting the number of files, number of commits, and number of lines of code of each pull request. Furthermore, we analyzed how the conjunction of these characteristics influences in the pull request lifetime. Section 5.4.2 presents the results of this analysis.

RQ 5.3 - Does the location of pull requests influence on their lifetime? For investigating characteristics about location of pull requests, we needed to conduct a within project analysis, given that files and directories possess a unique semantics for each project. Furthermore, we also carried out a qualitative analysis for understanding some of the observed patterns. Since the number of altered files and directories is large, we selected attributes using the InfoGain measure [31], which evaluates the importance of an attribute by verify the information gain associated with it. The ten most selective files and directories in relation to lifetime were chosen and used during association rule mining. Section 5.4.3 presents the results of this analysis.

RQ 5.4 - Does the profile of external contributors influence on the lifetime

of pull requests? We extracted rules in order to detect relationships between the number of previous contributions of the requesterexternal contributors experience with pull requests and lifetime. Complementarily, we explored rules involving physical attributes and requesters' lack of experience, aiming at establishing transitivity with lifetime. We performed an extra analysis to isolate the relationship between physical characteristics of contributions and external contributors profiles. The obtained results are discussed in Section 5.4.4.

RQ 5.5 - Do characteristics related to the review process influence on the lifetime of pull requests? Regarding the characteristics related to the review process of pull requests, besides mining rules involving lifetime, we extracted complementary rules to understand the relationship between physical characteristics and the comments made during the review process. We also conducted a qualitative analysis to subsidize the comprehension of social patterns related to reviewers of pull requests. In Section 5.4.5 we present the findings of this analysis.

We extracted 97,463 pull requests (excluding self-analyzed contributions) from 30 non-forked projects. From this total, 76.84% were accepted and 23.16% were rejected. Table 5.1 shows some characteristics of the analyzed projects: number of pull requests, acceptance rate, number of different requesters, number of pull requests submitted by core team members, and number of pull requests submitted by external members. The projects were randomly selected from 1,438 projects, based on a query that retrieved projects with large amounts (at least 500) of pull requests[1]. We collected pull requests sent from November 2010 to September 2015. We extracted rules with a minimum *support* of 1% (approximately 975 rules) to ensure that the rules were not obtained by chance, revealing a pattern from the dataset. In addition, we have defined a minimum confidence of 5%. We use these values to get as many rules as possible about the nature of the pull request paradigm.

As mentioned before, our studies we conducted experiments with most of the attributes extracted via GHTorrent. However, only a small set of them revealed rules with appropriate semantics in this scenario. The attributes considered in the analysis are presented in the Table 5.2.

[1]http://ghtorrent.org/dblite/

Table 5.1: Characteristics of the project corpus.

Project	**# Pull Requests**	**Acceptance Rate (%)**	**# Different Requesters**	**# PRs submitted by core team**	**# PRs submitted by external members**
Akka	1,020	84.0	255	85	936
Baystation12	5,196	96.0	208	219	4,977
Bitcoin	2,934	78.2	457	82	2,852
Boto	1,226	79.9	603	108	1,118
Brackets	3,823	91.6	377	173	3,650
Commcare-hq	6,986	98.8	50	20	6,966
Diaspora	1,746	76.6	321	78	1,668
Django	3,731	15.9	1,502	766	2,965
Docker	6,979	84.1	1,437	248	6,731
Infinispan	3,114	9.7	103	191	2,923
IPython	3,234	88.2	486	177	3,057
Jekyll	1,238	71.8	613	119	1,119
Joomla-cms	960	11.2	254	153	807
Katello	1,066	93.8	62	237	829
Kuma	2,873	91.8	115	42	2,831
Metasploit	4,535	84.9	485	534	4,001
Nancy	893	86.3	257	75	818
Netty	907	17.9	284	171	736
Node	510	0.2	209	214	296
Pandas	2,233	62.8	534	236	1,997
Phobos	2,744	94.7	243	90	2,654
Puppet	3,255	75.7	474	300	2,955
Rails	11,728	75.0	2,934	569	11,159
Rosdistro	5,745	97.4	345	49	5,696
Scala	1,033	88.9	80	23	1,010
Scikit-learn	1,018	72.9	379	146	872
Titanium	6,044	94.0	196	110	5,934
TrinityCore	2,173	61.1	466	95	2,078
Kodi	3,816	84.0	560	70	3,746
Zf2	4,705	48.5	908	300	4,405

5.4 Results and Discussion

In this Section, we report and discuss the obtained results and present the answers to the research questions. In Section 5.4.1, we discuss how time and acceptance are related in the context of the pull-based development paradigm. Section 5.4.2 discusses the influence of physical characteristics on lifetime. Section 5.4.3 presents results about the influence of the location of pull requests on their lifetime. Then, Section 5.4.4 presents results related to the influence of external developers characteristics on the lifetime of pull requests.

Table 5.2: Attributes used in this study.

Attribute	**Description**
comments_pull	the number of comments made during the review, where *"some comments"* = 2 to 10 comments; *"many comments"* = more than 10 comments.
commits_pull	the number of commits in the pull request, with values: *"1 commit"*; *"some commits"*: 2 to 4 commits; and *"many commits"*: more than 4 commits.
developer_type	whether the contributor is internal or external.
directory_names	represents a set of 10 attributes, where each is the name of a directory. The value *"true"* indicates that a file from that directory was changed, the value *"false"* indicates otherwise.
file_names	represents a set of 10 attributes, where each is the name of a file. The value *"true"* indicates that a file was changed, the value *"false"* indicates otherwise.
files_changed	the number of added, removed, or edited files, with values: *"1 file"*; *"some files"*: 2 to 4 files; and *"many files"*: more than 4 files.
first_pull	whether the pull request is the first filed by the contributor.
has_followers	whether the external contributor has followers on GitHub, takes *"true"* or *"false"*.
lifetime	amount of time between the submission and closure of a pull request, where *"very short"* = 1 day; *"short"* = 1 to 3 days; *"medium"* = 3 to 10 days; and *"lengthy"* = more than 10 days.
lines_changed	the number of lines of code manipulated, with values: *"1 line"*; *"some lines"*: 2 to 20 lines; and *"many lines"*: more than 20 lines.
previous_ contributions	the contributor's prior experience, considering the number of previous requests pull, with values: *"no contribution"*; *"some contributions"*: 1 to 6; and *"many contributions"*: more than 6.
reviewer_follow_ external_contributor	whether the reviewer follows the external contributor on GitHub.
reviewer_pull	the name of the developer responsible for reviewing the pull request.
status_pull	whether the pull request was *"accepted"* (merged) or *"rejected"* (closed).

Finally, in Section 5.4.5, we discuss the influence of the review process characteristics on the lifetime.

5.4.1 RQ 5.1 - Is there an effective relationship between the lifetime of pull requests and their acceptance?

Our study extracted association rules to identify the influence of time on the acceptance of pull requests. Figure 5.1 shows the *Lift* values for rules in which *lifetime* is the antecedent and *status_pull* is the consequent, so we can observe whether the time needed to evaluate pull requests is related to their acceptance. The results allow for the conclusion that pull requests with shorter lifetimes increase, though shyly, the chances of acceptance. For instance, when pull requests are analyzed within 1 day (very short) or within 1 to 3 days (short), the chances of acceptance increase in 11% (*Lift*=1.11) and 4% (*Lift*=1.04), respectively. On the other hand when lifetime is very short, the chances of rejection drop significantly, that is, in 35% (*Lift*=0.65).

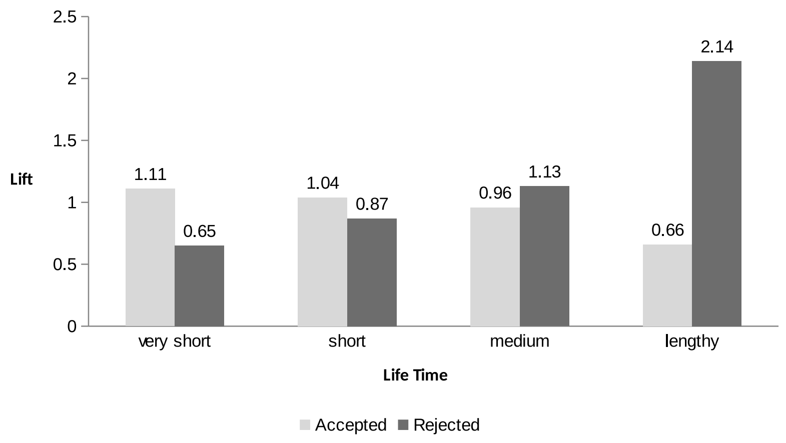


Figure 5.1: *Lifts* of the rules of type: *lifetime* $\rightarrow$ *status_pull.*

The results also indicate that contributions with lengthy lifetimes significantly increase the chances of rejection, and decrease the chances of acceptance. In our project corpus, when *lifetime* = *"lengthy"* occurs, the chances of rejection increase by 114% (*Lift*=2.14). In the same scenario, the chances of acceptance are reduced in 34% (*Lift*=0.66).

The results presented in this section indicate that there is a correlation between pull request lifetime and its acceptance/rejection. However, time is not a characteristic that can be easily controlled, that is, neither the contributor nor the reviewer can determine how long it will take for the pull request review process to complete. We believe that lifetime is a characteristic that can be determined by other characteristics related to the

RQ 5.1 - Is there an effective relationship between the lifetime of pull requests and their acceptance?
Answer - The results show that there is a correlation between pull request lifetime and acceptance/rejection, evidencing that contributions that have been quickly analyzed tend to be incorporated (chances increase in up to 11%), and the ones with lengthy lifetimes tend to be rejected (chances increase in up to 114%).
Relevance: Our results confirm the relationship of lifetime and the acceptance of pull requests as previously discussed by Gousios et al. [4] and Soares et al. [21]. In addition, our results quantify the strength of such relationship by means of measures such as *Confidence* and *Lift* .
Yu et al. [9] did not answer this question.
Implications: Our results provide initial evidences that lifetime leads to acceptance/ rejection, offering insights into the importance of a fast review process.

context of the pull request paradigm, motivating the existence of a common cause of both lifetime and acceptance. With that being said, Sections 5.4.2, 5.4.3, 5.4.4, and 5.4.5 present characteristics that can be determinants of lifetime.

5.4.2 RQ 5.2 - Do physical characteristics of pull requests influence on their lifetime?

Pull requests possess some physical characteristics, such as the number of modified files, number of commits, and number of changed lines of code. These characteristics are known in the moment of the submission of a pull request, but can undergo further modifications during its lifetime. For instance, the discussion about a contribution may demand new commits. In this subsection, we discuss the influence of these characteristics on the lifetime of pull requests.

Our results indicate that the number of commits associated with a pull request influences the lifetime of contributions. Figure 5.2 displays the *Lifts* values of rules where *commits_pull* is the antecedent and $lifetime$ is the consequent. For the contributions with many commits, the chances for pull requests with lengthy lifetime increase significantly. This pattern becomes utterly evident when we analyze the rule *commits_pull = "many commits"* $\rightarrow$ *lifetime = "lengthy"*. In this case, when there are many commits in a contribution, the chances of a lengthy lifetime increase in 92% (*Lift*=1.92).

These results show that there is a correlation between *commits* and $lifetime$. Considering that the number of commits can increase throughout a pull request's lifetime, it is important to verify whether the increase in the amount of commits is an implication of the time taken for review. In order to verify that, we carried out a *confidence*

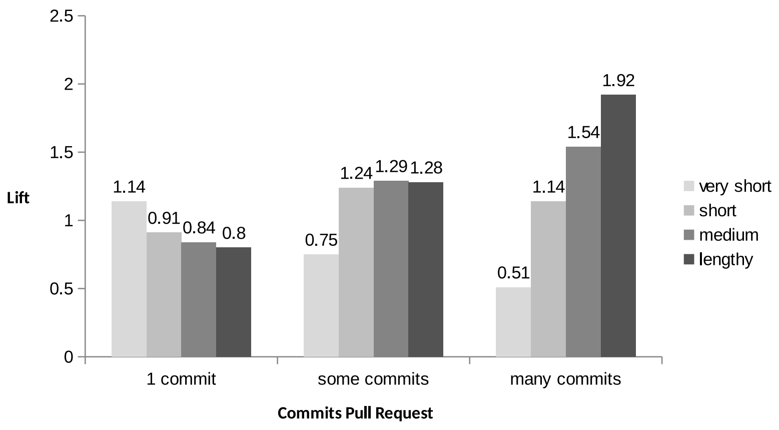


Figure 5.2: *Lifts* of the rules of type: *commits_pull* $\rightarrow$ *lifetime.*

analysis of rules that involve these two attributes. The *confidence* of the rule *commits_pull = "many commits"* $\rightarrow$ *lifetime = "lengthy"* is 32%, while the *confidence* of its inverse, *lifetime = "lengthy"* $\rightarrow$ *commits_pull= "many commits"*, is 17%. Therefore, the amount of commits seems not to be an implication of the time taken for review process. These analyses provide some initial evidence that the complexity required for the review process of pull requests is a characteristic that impacts the lifetime of contributions.

Similarly to commits, the number of modified files in a pull request represents another factor that impacts the lifetime. Figure 5.3 displays the *Lift* values for rules of type $files\_changed \rightarrow lifetime$. It is observable that smaller contributions in terms of number of files increase the chances of very short lifetimes. For instance, pull requests which alter only 1 file have 21% more chances of being revised within up to one day (*Lift=1.21*). On the other hand, contributions that modify various files have 54% more chances of having a longer lifetime (*Lift=1.54*).

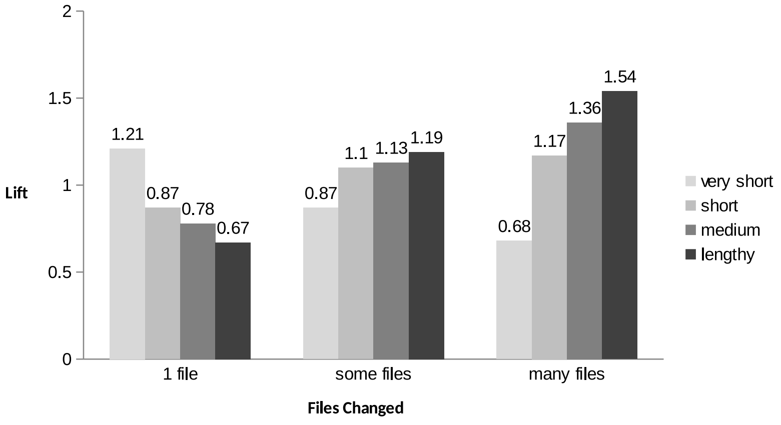


Figure 5.3: *Lifts* of the rules of type: *files_changed* $\rightarrow$ *lifetime.*

The results also suggest that there is a relationship between the number of modified lines of code in a pull request and lifetime, especially when several lines of code have been altered. According to Figure 5.4, pull requests that modify several lines of code have 43% more chances of having lengthy lifetime (*Lift*=1.43). Notwithstanding, there are no means to establish that many lines effectively implies an increase in lifetime, since the *confidence* of the rule *lines_changed = "many lines " → lifetime = "lengthy"* is 24%, while the *confidence* of its inverse, *lifetime = "lengthy" → lines_changed = "many lines"*, is 67%. This analysis suggests that there is a correlation between number of modified lines of code and lifetime: 67% of the pull requests that have lengthy lifetimes also have several modified lines, but only 24% of the contributions with several modified lines have lengthy lifetime.

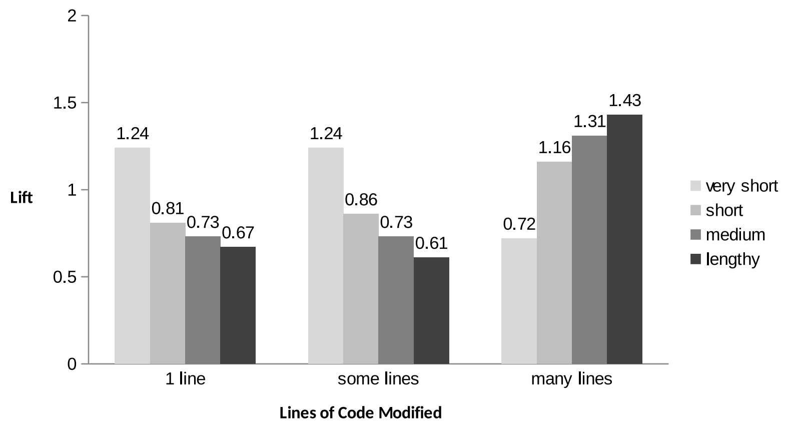


Figure 5.4: *Lifts* of the rules of type: *lines_changed → lifetime.*

Our results also allow the indication that the conjunction of physical characteristics may increase the chances of pull requests having very short and lengthy lifetimes. Table 5.3 presents rules that combine different physical characteristics in the antecedent, as well as their measures of interest. Rule 1, for example, shows that the chances of a very short lifetime increase in 35% (*Lift*=1.35) when the pull request is composed of one commit, changing between 2 and 20 lines of code in one file. This reinforces that simpler pull requests contribute for a lifetime of up to one day. It is noteworthy that, when occurring alone, none of these three characteristics leads to elevated *Lift* value. On the other hand, the analysis of Rule 2 suggests that the occurrence of just two characteristics together can ensure a very short lifetime. In this case, when a contribution has merely one commit changing just one file, the chances of it being accepted or rejected within up to 24 hours increase in 29% (*Lift*=1.29). Rules 1 and 2 suggest desirable characteristics in order for a pull request to be analyzed readily. In this analysis, we did not extract the rules with *lines_changed = "1 line"* in the antecedent, because they have *support* below the minimum and do not represent a significant pattern in the dataset.

Table 5.3: Association rules involving physical characteristics of pull requests relevant to lifetime.

#	Antecedent	Consequent	*Sup* (%)	*Conf* (%)	*Lift*
1	*commits_pull = "1 commit"* ∧ *files_changed = "1 file"* ∧ *lines_changed = "some lines"*	*lifetime = "very short"*	23.0	70.0	1.35
2	*commits_pull = "1 commit"* ∧ *files_changed = "1 file"*	*lifetime = "very short"*	28.0	67.0	1.29
3	*commits_pull = "many commits"* ∧ *files_changed = "some files"* ∧ *lines_changed = "many lines"*	*lifetime = "lengthy"*	1.0	37.0	2.25
4	*commits_pull = "many commits"* ∧ *files_changed = " many files"* ∧ *lines_changed = "many lines"*	*lifetime = "lengthy"*	2.0	30.0	1.79

Rule 3 from Table 5.3 shows under which conditions the conjunction of physical characteristics implies a lengthy lifetime. When pull requests have many commits changing many lines of code of some files, they have 125% more chances of having lengthy lifetime (*Lift*=2.25). Despite *"many_files"* leading to higher than *"some lines" Lift* value when occurring individually (*Lift*=1.54) – see Figure 5.3, when this characteristic is associated with *"many_commits"* and *"many_lines"*, the chances of slowness increase by just 79% (*Lift*=1.79), as indicated by Rule 4 from Table 5.3. Rules 3 and 4 indicate physical characteristics that should be avoided for a faster processing of a pull request.

RQ 5.2 - Do physical characteristics of pull requests influence on their lifetime?
Answer - Physical characteristics such as number of commits, number of files, and number of lines of code influence on the lifetime of pull requests, both in an isolated and in a conjoint fashion.
Relevance: Our results confirm the influence of the number of commits, as previously discussed by Yu et al. [9]. In addition, our results quantify the strength of such influence by means of measures such as *support*, *confidence*, and *Lift*. Moreover, we also identified the influence of other attributes, such as number of files and number of lines of code, on the lifetime of pull requests. Finally, we could also observe the joint influence of such attributes in the lifetime. Gousios et al. [4] and Soares et al. [21] did not explored this question.
Implications: Based on our results, requesters can adopt practices that reduce the complexity of their contributions so that they have a short lifetime and, consequently, increase the chances of acceptance. On the other hand, project managers can encourage these practices in their contribution policies.

5.4.3 RQ 5.3 - Does the location of pull requests influence on their lifetime?

As mentioned earlier, pull requests change files in software repositories that, in general, are organized in a hierarchical directory structure. Consequently, the directory and file indicate the change location. This way, through the extraction of data from pull requests, it is possible to identify which files and directories are changed by a contribution, knowing exactly the pull request location.

Aiming at understanding the influence of pull request location on lifetime, we extracted association rules from the TrinityCore repository[2] using the attributes *file_names* and *directory_names*. Even though pull request location is not purely determined by the external contributor, but by the need of the contribution itself, the extracted rules reveal that, depending on which artifacts are manipulated and where they are stored, the contributions may lead to different lifetimes, as shown in Table 5.4. It is important to observe that, as location is project dependent (same files and directories usually do not occur in different projects), this analysis was made to a specific project: TrinityCore.

Table 5.4: Association rules involving pull requests locality in relation to lifetime *"very short"*.

#	**Antecedent**	***Sup*** (%)	***Conf*** (%)	***Lift***
1	*file_names = ".../smartscripts/smartai.cpp"*	2.0	75.0	1.35
2	*file_names = ".../smartscripts/smartscriptmgr.cpp"*	2.0	83.0	1.50
3	*file_names = ".../smartscripts/smartai.cpp"* ∧ *file_names = ".../smartscripts/smartscriptmgr.cpp"*	1.0	96.0	1.72
4	*directory_names = "...updates/world"*	14.0	49.0	0.89
5	*directory_names = "...tools/mesh_extractor"*	1.0	96.0	1.73
6	*directory_names = "...updates/world"* ∧ *directory_names = "...tools/mesh_extractor"*	1.0	100.0	1.80

Rules 1 and 2 indicate that pull requests which manipulate the file *".../smartscripts/smartai.cpp"* or *".../smartscripts/smartscriptmgr.cpp"* have 35% *(Lift=1.35)* and 50% *(Lift*=1.50) more chance of having very short lifetimes, respectively. Rule 3 evidences that, when these files are modified together in the same pull request, the chances of having very short lifetimes increase in 72% (*Lift*=1.72). Besides being stored in the same directory, a qualitative analysis of the software repository revealed logic coupling between these artifacts. These rules may indicate that, for instance, some contributions may be prioritized by the core team, depending on the files changed by the pull request.

[2]https://github.com/TrinityCore/

The results also suggest that the directories to which the files belong have influence on lifetime. Rules 4, 5, and 6 demonstrate this implication. Rule 4, for instance, indicates that contributions that manipulate files from the directory *"...updates/world"* have 11% less chance of having very short lifetime (*Lift*=0.89). Rule 5 shows that when a contribution modifies files in the directory *"...tools/mesh_ extractor"*, the chances of very short lifetime increase in 73% (*Lift*=1.73). Rule 6 points out that, though *"...updates/world"* decreases the chances of very short lifetime, when pull requests manipulate files from this directory together with files from directory *"...tools/mesh_ extractor"*, the contributions are always accepted or rejected within up to 24 hours (*Confidence*=100%). Furthermore, this rule implies that the antecedent increases in 80% the chances of occurrence of the consequent (*Lift*=1.80).

Although we have analyzed the influence of location in only one project, we were careful to statistically verify if the observed influence actually exists in other projects. We randomly selected 20% of the projects of our data set, and for each selected project, we identified, via attribute selection (using the InfoGain measure), the top 10 files related to the class (lifetime), and extracted rules of the type *file → lifetime*. In all projects, most of the selected files indicated negative or positive influence for one of the labels (*"very short"* and *"lengthy"*), individually. Table 5.5 displays *Lift* values for rules extracted from the Bitcoin, Django, IPython, Pandas, Rails, and ZF2 projects. For example, in Bitcoin project, there are files that increase the chances of lifetime very short in up to 143% (*Lift*=1.43), while other files decrease the chances in up to 24% (*Lift*=0.76).

Table 5.5: *Lifts* of the rules of type: *files_names → lifetime = "very short"(VS) or "lengthy"(L).*

Index	**Projects**											
	Bitcoin		**Django**		**IPython**		**Pandas**		**Rails**		**ZF2**	
	VS	**L**	**VS**	**L**	**VS**	**L**	**VS**	**L**	**VS**	**L**	**VS**	**L**
1	2.43	0.28	0.66	1.66	0.69	1.51	0.71	1.29	0.73	1.51	0.82	1.44
2	1.68	0.24	0.34	2.01	0.43	2.87	0.45	2.99	0.56	1.54	1.68	0.90
3	0.74	1.43	0.38	1.74	0.38	2.36	0.49	1.50	0.64	2.23	0.71	1.59
4	0.68	1.49	0.46	1.50	0.38	1.77	0.36	1.46	0.60	2.12	0.97	1.04
5	0.68	1.64	0.58	1.75	0.53	1.99	0.37	1.75	0.76	1.78	0.60	2.92
6	0.76	1.77	0.46	1.84	0.39	2.04	0.56	1.04	0.70	1.80	0.44	1.75
7	0.65	1.70	0.35	2.04	0.39	2.07	0.49	1.58	0.66	1.56	0.68	1.65
8	0.39	1.98	0.30	2.23	0.38	1.80	0.56	1.65	0.71	2.00	0.52	0.82
9	0.64	1.85	0.23	1.99	0.36	1.77	0.52	1.74	0.54	2.55	0.41	1.64
10	0.76	1.60	0.92	1.38	0.56	1.84	0.63	1.44	0.72	1.77	0.76	0.50

RQ 5.3 - Does the location of pull requests influence on their lifetime?
Answer - The files changed by the contributions and their directories may increase the chances of very short lifetime.
Relevance: This was an open question until now, as none of the related work provided answers to it. As far as we know, our study was the first to provide evidences of the influence of location on the lifetime.
Implications: This knowledge provides insights to requesters about areas that tend to have pull requests with lengthy lifetime and, consequently, reduce chances of acceptance. Therefore, they may decide to avoid codebase regions sensitive to changes in order reduce chances of slowness.

5.4.4 RQ 5.4 - Does the profile of external contributors influence on the lifetime of pull requests?

The analysis of the extracted association rules suggests that the external contributors lack of experience with the collaboration paradigm influences on the lifetime of the contributions. Figure 5.5 shows *Lift* values for the rules in which *first_pull* is the antecedent and *lifetime* is the consequent. The first pull request of a developer usually leads to a lengthy lifetime. When a developer's first contribution occurs (*first_pull* = *"true"*), the chances of it having lengthy lifetime increase in 85% (*Lift*=1.85), while also reducing the chances of very short or short lifetime in 24% (*Lift*=0.76) and 13% (*Lift*=0.87), respectively. On the other hand, when this is not the case (*first_pull* = *"false"*), the pull request has 23% (*Lift*=0.77) less chances of being lengthy. However, this fact (*first_pull* = *"false"*), individually, does not imply a significant increment in the chances of faster lifetimes. With that being said, we can draw the conclusion that the utilization of the collaboration paradigm for the first time raises the probability of lengthy lifetimes.

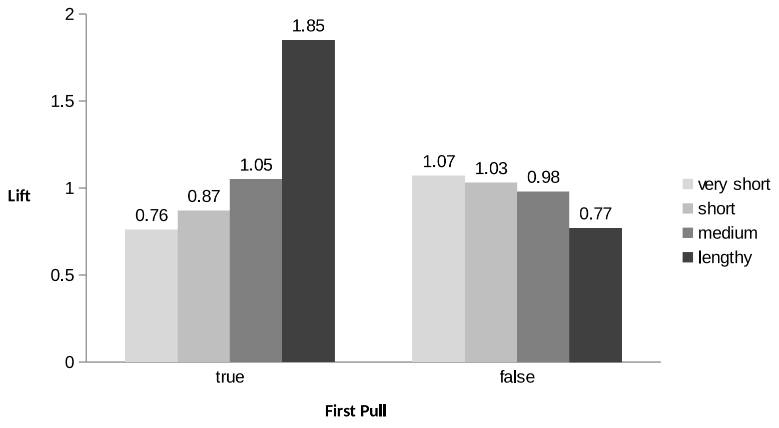


Figure 5.5: *Lifts* of the rules of type: *first_pull* → *lifetime.*

Moreover, the results show that should a developer previously made a large amount of contributions, the chances of slowness decreases. Figure 5.6 displays *Lift* values of rules in which *previous_ contributions* is the antecedent and *lifetime* is the consequent. Even though the submission of many pull requests does not imply a significant increment in the chances of shorter lifetimes, when this occurs, there is a significant decrease in the probability of slowness, namely 30% (*Lift*=0.70).

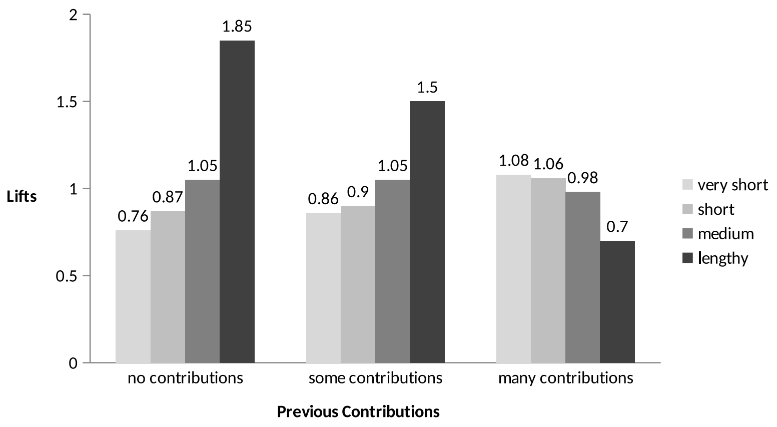


Figure 5.6: *Lifts* of the rules of type: *previous_ contributions* $\rightarrow$ *lifetime*.

Our results allow the conclusion that the developer's history of contributions via pull requests represents a characteristic of strong influence on lifetime. Therefore, it is possible to remind developers that their frequency in the submission of pull requests may result in a decrease of the lifetime of their contributions.

Additionally, we decided to investigate the quality of the first pull request in terms of physical characteristics that usually lead to lengthy lifetime. To do so, we extracted rules with *first_pull* = *"true"* as the antecedent and some physical characteristic as the consequent. The rules illustrated in Table 5.6 show that *first_pull* = *"true"* does not significantly increase the chances of occurrence of physical characteristics that imply complexity in the review, instead, the *confidence* values indicate that 15% of the pull requests manipulated several files and only 9% possessed many commits associated with them. Thus, we conclude that developers first contributions tend to slowness due to nonphysical characteristics, meaning that the causes may be technical or social.

Another characteristic that appears to have impact on lifetime is the type of developer (i.e., *core_ team* or *external*) who submits pull requests. According to [5], in general, pull requests are performed by external members. However, core team developers also use them as a tool for stimulating discussions prior to their integration in the repository.

Figure 5.7 displays *Lift* values of rules of type *developer_ type* $\rightarrow$ *lifetime*. The results

Table 5.6: Association rules involving *first_pull* and some physical characteristics of pull requests.

#	Antecedent	Consequent	*Sup* (%)	*Conf* (%)	*Lift*
1	*first_pull = "true"*	*commits_pull = "1 commit"*	14.0	69.0	1.00
2		*commits_pull = "some commits"*	4.0	22.0	1.00
3		*commits_pull = "many commits"*	2.0	9.0	1.04
4		*files_changed = "1 file"*	11.0	52.0	1.06
5		*files_changed = "some files"*	7.0	33.0	1.02
6		*files_changed = "many files"*	3.0	15.0	0.81
7		*lines_changed = "1 line"*	1.0	4.0	1.15
8		*lines_changed = "some lines"*	11.0	53.0	1.07
9		*lines_changed = "many lines"*	9.0	43.0	0.91

evidence that pull requests submitted by external members have virtually no influence on lifetime – *Lift* values nearing conditional independence. However, pull requests submitted by core team members present strong influence on the slowness of lifetime – preliminary analysis revealed that 70.93% of pull requests submitted by *core_team* have lifetime exceeding 3 days and 36.13% more than 10 days. When a pull request is submitted by core team members, the chances of having *lifetime= "medium"* increase in 26% (*Lift*=1.26), and the chances of having *lifetime = "lengthy"* increase in 118% (*Lift*=2.18). Complementarily, in order to ensure the existence of this pattern, we verified that the *confidence* of the rule *developer_type = "core_team"* $\rightarrow$ *lifetime = "lengthy"* is of 38%, while the *confidence* of the inverse rule is merely 13%. This allows us to conclude that a lengthy lifetime may be influenced by *developer_type = "core_team"*, as one could expect, instead of time influencing the type of developer.

Except for the cases where the project policy requires, core team members are not obligated to send their contributions via pull requests, as they have write privileges to perform direct changes into the repository. That said, there is, additionally, a need to identify patterns present in the contributions submitted by core team members that can aid in the understanding of the causes of lengthy lifetimes. To that end, we extracted association rules in which the attribute *developer_type = "core_team"* is the antecedent and the consequent is composed of physical and social characteristics. Rules 1, 2, and 3 from Table 5.7 show that when *developer_type = "core_team"*, the chances of having many commits, several files, and many lines increase in 125% (*Lift*=2.25), 27% (*Lift*=1.27), and 36% (*Lift*=1.36), respectively. Thus, we are able to conclude that the submissions made by core team members are more complex (from a physical standpoint) hence demanding more time for analysis.

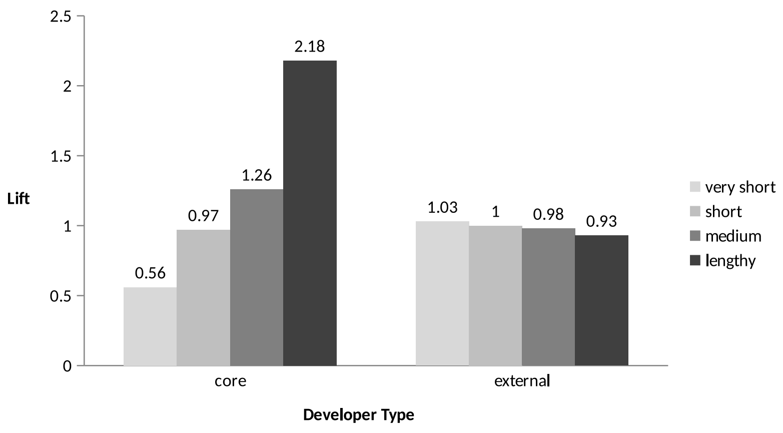


Figure 5.7: *Lifts* of the rules of type: *developer_type* $\rightarrow$ *lifetime.*

Table 5.7: Association rules involving *core_team members*, and physical and social characteristics.

#	Antecedent	Consequent	*Sup* (%)	*Conf* (%)	*Lift*
1	*developer_type = core_team*	*commits_pull = "many commits"*	1.0	19.0	2.25
2		*files_changed = "many files"*	1.0	24.0	1.27
3		*lines_changes = "many lines"*	4.0	64.0	1.36
4		*comments_pull = "no comments"*	3.0	53.0	0.70
5		*comments_pull = "many comments"*	1.0	18.0	2.29

When we analyze characteristics associated with the discussions during the assessment of pull requests, it is possible to observe that contributions submitted by members of the core team tend to have more comments. Rule 4 in Table 5.7 indicates that when internal members submit pull requests, the chances of a review without comments are reduced in 30% (*Lift*=0.70), while Rule 5 evidences that the likelihood of having many comments is 129% (*Lift*=2.29) higher when *developer_type = "core_team"*. This way, it is possible to infer that internal contributions are more liable to discussion.

Hosting tools like GitHub provides social network resources, allowing developers to have "followers" and a social relationship among themselves. We also investigated whether social characteristics regarding the external contributor may influence on lifetime.

Initially, we analyzed whether the fact that external contributor have or not followers would be the root cause for the time spent evaluating a pull request. Figure 5.8 shows *Lift* values of rules of type *has_followers* $\rightarrow$ *lifetime.* Thus, we can observe that the social characteristic of having followers (*has_followers = "true"*) does not influences on lifetime – *Lift* values close to 1 indicate conditional independence. However, when the

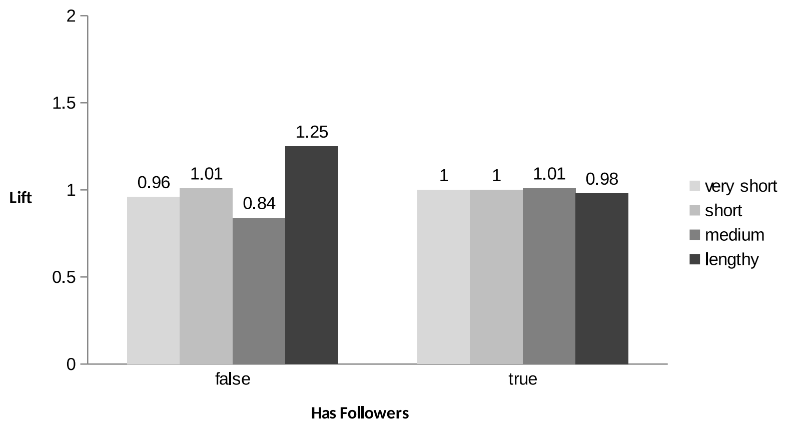


Figure 5.8: *Lifts* of the rules of type: *has_followers* → *lifetime*.

external contributor does not have followers (*has_followers* = *"false"*) the chances of lengthy lifetime increase in 25% (*Lift*=1.25).

We can affirm that when a developer follows someone on GitHub, there is at least some relationship that indicates interest on the activities of the other developer. For example, if Peter follows John, Peter knows or has interest in knowing John's work; however, the contrary is not always true. Based on this, we analyzed another characteristic inherent to the external contributor. We extracted association rules with the attribute *reviewer_follow_external_contributor* in the antecedent — indicating whether the core team member responsible for the review follows the author of the pull request — and lifetime in the consequent. Figure 5.9 shows the *Lift* values for these rules and allows the conclusion that *reviewer_follow_external_contributor* = *"false"* does not influence the lifetime, however when *reviewer_follow_external_contributor* = *"true"* the chances of slowness are reduced in 34% (*Lift*=0.66). Therefore, there are less chances of lengthy lifetimes when the reviewer follows the external contributor, indicating that the interest of the former for the work of the latter reduces the chances of slowness.

5.4.5 RQ 5.5 - Do characteristics related to the review process influence on the lifetime of pull requests?

GitHub provides a discussion forum for each pull request, enabling developers to interact through comments. The number of comments may increase during the review of pull requests, and, to some extent, influence on the lifetime. Our results indicate that contributions without comments are analyzed very quickly, despite the existence of some exceptions. Complementary, pull requests with several comments tend to be lengthy, even

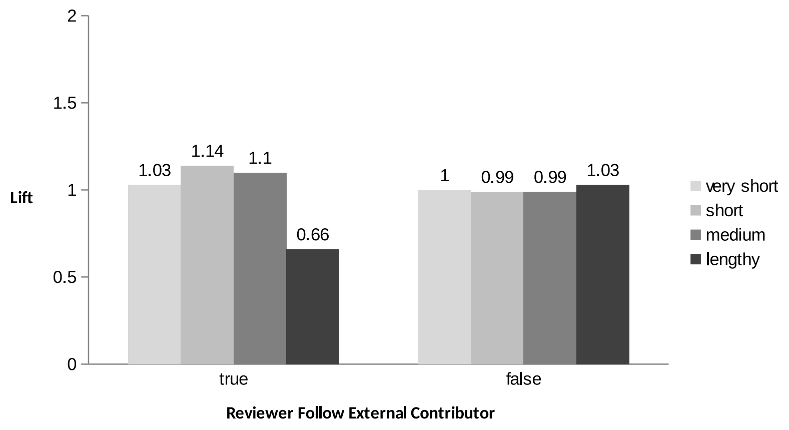


Figure 5.9: *Lifts* of the rules of type: *reviewer_follow_ external_ contributor* $\rightarrow$ *lifetime.*

RQ 5.4 - Does the profile of external contributors influence on the lifetime of pull requests?

Answer - Characteristics associated with the external contributor, such as the number of previous contributions, his or her profile as a developer, and his or her social relations have impact on lifetime.

Relevance: Our results confirm that pull requests filed by people with many previous contributions have shorter lifetime, as previously discussed by Gousios et al. [4] and Yu et al. [9]. Conversely, our study provides evidences that pull requests filed by core team members tend to have lengthy lifetimes. This finding, contrast with the results discussed by Gousios et al. [4] and Yu et al. [9]. Soares et al. [21] did not explored this question.

Implications: The results are potentially useful for project managers to adopt strategies that improve the pull requests review process, for example, by properly assigning core team members to review internal contributions and reducing the lifetime.

though there are also some exceptions.

Figure 5.10 displays the *Lift* values of rules of type *comments_pull* $\rightarrow$ *lifetime* and allows a more detailed analysis of the influence of comments on lifetime. The existence of no comments during the review of pull requests increase in 17% the chances of having *lifetime* = *"very short"* (*Lift*=1.17) and reduces the chances of lifetime being longer than 24 hours (*Lifts* values ranging from 0.72 to 0.93). The existence of only one or few comments seems to lower the chances of having *lifetime* = *"very short"* (*Lift* values equal to 0.69 and 0.56, respectively). The occurrence of many comments leads to an opposite behavior of *comments_pull* = *"no comments"*, but much more intense. When *comments_pull* = *"many comments"* occurs, the chances of lifetime being very fast decreased in 78% (*Lift*=0.22), while the chances of slowness are raised in 162% (*Lift*=2.62).

Comments on pull requests occur due to various reasons. There are comments, for

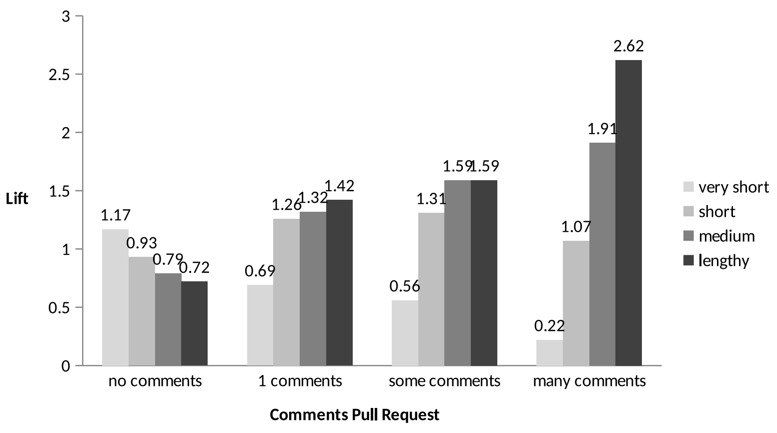


Figure 5.10: *Lifts* of the rules of type: *comments_pull* $\rightarrow$ *lifetime.*

example, where the reviewer thanks the external contributor for his or her contribution, poses a question, suggests a modification, etc. Even though the influence of comments on lifetime is evident, it is noteworthy that, similarly to lifetime, the number of comments can be determined by other characteristics. This research does not aim at specifically identifying these characteristics, but we present rules that showcase the relationship between comments and some physical characteristics — such as number of commits and changed files, in Table 5.8. Rules 1 and 2, for instance, reveal that pull requests without comments tend to be simpler, considering number of commits and changed files, although the evidence is weak (*Lifts* of 1.10 and 1.14, respectively). On the other hand, contributions that have several comments tend to be more complex, as evidenced by Rules 3 and 4. The *Lift* values of 3.60 and 2.53 point out a significant increase in the chances of pull requests with many comments having many commits and changed files, respectively.

Table 5.8: Association rules involving *comments_pull* and physical characteristics.

#	**Antecedent**	**Consequent**	***Sup*** (%)	***Conf*** (%)	***Lift***
1	*comments_pull = "no comments"*	*commits_pull = "1 commit"*	58.0	76.0	1.10
2		*files_changed = "1 file"*	42.0	56.0	1.14
3	*comments_pull = "many comments"*	*commits_pull = "many commits"*	3.0	31.0	3.60
4		*files_changed = "many files"*	4.0	47.0	2.53

Pull requests must be closed – accepted or rejected – by core team members, who may be asked to do so or who pick the pull request spontaneously. This is inherently connected with the review/discussion process of the project. We extracted pull requests from the TrinityCore repository in order to identify whether the developer responsible for the review may impact the lifetime of pull requests.

The results point out that, depending on which developer analyzes the pull request, there may be impact on lifetime. Table 5.9 displays rules in which *reviewer_pull* is the antecedent and *lifetime* the consequent. Rules 1 and 2 evidence cases where the chances of very short lifetime increase when the pull requests are analyzed by certain reviewers. Although the *Lift* values already indicate this, the *confidence* of the rules reinforce this pattern. Rule 1 shows that, when *Discover* is the one responsible for review, lifetime is up to one day long in 80% of the cases; besides, the chances of a very short lifetime increase in 44% (*Lift*=1.44). Rule 2 unveils a similar scenario. Nonetheless, there are reviewers that tend to slowness, thus featuring longer lifetimes. Rules 3, 4, and 5 illustrate this pattern. For instance, when *Jackpoz*, *MitchesD*, and *Joschwald* are responsible for analyzing the pull request, the chances of slowness increase in 151% (*Lift*=2.51), 81% (*Lift*=1.81), and 67% (*Lift*=1.67), respectively.

Table 5.9: Association rules involving reviewers and lifetime.

#	**Antecedent**	**Consequent**	***Sup* (%)**	***Conf* (%)**	***Lift***
1	*reviewer_pull = "Discover"*	*lifetime = "very short"*	2.0	80.0	1.44
2	*reviewer_pull = "Shauren"*		8.0	72.0	1.30
3	*reviewer_pull = "Jackpoz"*	*lifetime = "lengthy"*	2.0	45.0	2.51
4	*reviewer_pull = "MitchesD"*		1.0	32.0	1.81
5	*reviewer_pull = "Joschwald"*		1.0	30.0	1.67

These results indicating that reviewers have impact on lifetime motivate a qualitative analysis. We conceived an analysis based on characteristics knowingly influential regarding the lifetime. When analyzing pull requests evaluated by the developer *Jackpoz*, for example, we found no evidence that physical characteristics were the cause of the slowness of the review. Out of 37 pull requests analyzed by this developer with *lifetime = "lengthy"*, only 29.7% of them had more than one comment, 24.3% had more than one commit, and only 18.9% manipulated several files. This shows intrinsic characteristics of Jackpoz.

We selected another developer for a qualitative analysis. We analyzed all of *MitchesD*'s contributions which had lengthy lifetimes. Out of 17 pull requests closed by *MitchesD*, 4 had more than 10 comments and 11 had between 2 and 10 comments. Although comments imply slowness, it is not always true that *MitchesD* was the root cause of the slowness. Our analysis revealed that, in several contributions evaluated by him, there was barely any discussion motivated by him. In many cases, *MitchesD* closed contributions without interacting with other developers.

The results indicate, therefore, that some developers may have influence on the lifetime of pull requests due to intrinsic characteristics of the review process (*Jackpoz*'s case). However, there are cases where what seems to be a malefic pattern is, in fact, a core team member's well-intentioned action (*MitchesD*'s case) that closed long-standing contributions still open and without definition.

RQ 5.5 - Do characteristics related to the review process influence on the lifetime of pull requests?
Answer - Social characteristics regarding pull request review have influence on lifetime. Characteristics such as number of comments may influence the chances of slowness or fastness of the review process. The developer who is responsible for the review may also have influence on lifetime.
Relevance: Our results confirm the influence of the number of comments on the lifetime, as previously discussed by Yu et al. [9]. In addition, our results quantify the strength of such influence by means of measures such as *support*, *confidence*, and *Lift*. Moreover, we also found evidences that the reviewer influences on the lifetime. Gousios et al. [4] and Soares et al. [21] did not explored this question.
Implications: Our findings show to project managers the need to appropriately assign pull requests reviewers to reduce lifetime, increasing core team efficiency, since time saved frees core team members to other activities.

5.5 Threats to Validity

Even though we have worked carefully to minimize threats to validity of this study, some uncontrolled factors might have influenced the observed results. Regarding the correctness of our results, our corpus, composed of 30 projects, led to some association rules with low *support*, approximately 1%, which represents 975 instances. Nonetheless, we discarded the possibility that these patterns might have arisen accidentally, as we identified a set of patterns whose frequent occurrence proves difficult to be assumed by chance. Even so, we acknowledge the necessity to corroborate all of our results with a wider approach, involving a multitude of projects.

We also advert that the capability of generalizing our observations is restricted to projects that possess the same features as the ones we have selected: all of them are open-source, the majority of contributions is accepted quickly, and the projects have certain maturity with the utilization of pull requests. Therefore, we cannot generalize our results for industrial or open-source projects with differing features, and not even guarantee that the patterns revealed by our study are valid for new projects, from a collaboration model standpoint. Our analysis, nevertheless, revealed that similar patterns may be present in other projects. Further future studies have to be conducted in order to assess this.

5.6 Final Remarks

In this chapter, we presented a study on the characteristics that impact the lifetime of pull requests in open-source projects with a large number of contributions. We also considered that pull request lifetime is ruled by characteristics inherent to the pull requests and their review process. The knowledge obtained from our results unveils patterns that can be followed or avoided, both by core team members and external contributor, in order to promote positive scenarios or weaken the negative ones that have impact on the pull request lifetime. Project managers, for instance, are often interested in the efficiency of the team and the celerity of the analysis of external contributions, since this reduces the amount of time spent for adding new features or fixing bugs in the product. For similar reasons, knowing what causes the fast acceptance or rejection of a pull request is definitely of interest to whom collaborates with a project. Thus, in this sense, we perceive pull request lifetime as a matter of crucial importance.

Due to the extraction of association rules, our study was able to quantify the degree of influence of the investigated characteristics regarding the duration of pull request lifetime. In order to complement this quantitative analysis, we also conducted a qualitative analysis over some of the detected patterns. In particular, the results allowed the following conclusions: (i) there is an effective relationship between pull request lifetime and their acceptance, in which contributions with shorter lifetimes tend to have the chances of being incorporated into the main repository increased, while the slower ones tend to have the chances of being rejected increased; (ii) physical characteristics such as number of commits, altered files, and changed lines of code influence, in an isolated or combined way, the pull request lifetime; (iii) the artifacts modified by pull requests and the directories where they are stored can be robust predictors of the duration of pull request lifetime; (iv) characteristics regarding the external contributor, such as the number of previous contributions of the requester, his or her profile as a developer, and his or her social relationships have influence on lifetime; and (v) the amount of comments in a pull request, as well as the developer responsible for the review, are important predictors of lifetime duration.

Chapter 6

Influential Factors on Assigning Reviewers to Pull Requests in Open-Source Projects

6.1 Introduction

The process of assigning a reviewer to analyze a pull request is not always well defined. For example, the decision to review a given pull request may come directly from the core team members with interest in the pull request, potentially incurring in inappropriate assignments. Recently, a series of studies arose [10, 11, 12, 13, 14, 36], aiming at recommending reviewers to analyze pull requests. In general, these studies employ predictive techniques for suggesting reviewers based on the project history. Nonetheless, these recommendations mostly replicate previous (not always desired) patterns and work as black boxes – i.e., they do not uncover the reasoning behind the recommendation. Besides, the quality of the recommendation is tightly related to the chosen predictive attributes.

Understanding the factors that influence on assigning reviewers to pull requests is crucial to better design recommendation tools and provide adequate support for project managers. These factors may reveal possible conflicts of interest among requesters and reviewers, unbalanced workloads among reviewers, and regions of the codebase with few experts, representing a risk in face of turnover. Replicating such problems in future recommendation is clearly not desired. Comprehending the nature of reviewers assignment for analyzing pull requests is a first step for identifying these problems and quantifying how frequently they occur.

In this work, we mined association rules from 22,523 pull requests to identify factors that have an influence on the assignment of pull request reviewers. More specifically, we answer the following research questions:

RQ1 – Do the pull request characteristics influence on the reviewer assignment?

RQ2 - Does the requester profile influence on the reviewer assignment?

RQ3 - Does the social relationship between requester and reviewer influence on the reviewer assignment?

RQ4 - Does the location of a pull request influence on the reviewer assignment?

We could unveil the following patterns, among others: (i) some reviewers always analyze simple pull requests, with few commits and few files; (ii) some reviewers frequently (up to 58% of the times) analyze pull requests filed by inexperienced requesters; (iii) some reviewers have more chances (up to 25 times) to analyze pull requests filed by requesters of their acquaintance; and (iv) some reviewers have more chances (up to 20 times) to analyze pull requests containing files that they have recently changed. Besides the quantitative analysis using association rules mining, we also run a qualitative analysis, aiming at better understanding some of the uncovered patterns.

The remainder of this chapter is organized as follows. Section 6.2 presents the main works related to this study. Next, Section 6.3 presents the research process adopted by our study and the dataset used in the experiments. Section 6.4 shows the obtained results, answering each of the research questions and discussing the implications of our answers. Section 6.5 presents the threats to validity of our results. Finally, Section 6.6 concludes this work by summarizing our main contributions and pointing out some future work.

6.2 Related Work

As far as we are aware, previous work does not investigated the factors that influence on assigning reviewers to pull requests. Thus, our study is the first to bring forth results in this direction. Nevertheless, some research propose approaches for the recommendation of reviewers. These approaches are discussed in the following paragraphs.

Yu et al. [10] recommended reviewers to pull requests by means of techniques such as information Retrieval (IR) and Vector Space Model (VSM), measuring, based on the titles and descriptions of previous pull requests, the semantic similarity with the new pull request. Then, using classification, they foresee the adequateness of a given developer for reviewing a pull request according to the number of comments that he or she may have previously made in the review process. Besides, the approach proposes a network of comments for each project, analyzing the relationship of comments made by developers. Therefore, they can predict the common interest of each reviewer with the requester. The approach was evaluated over 10 popular projects from GitHub that had received more than 1,000 pull requests. The results achieved a *precision* of 74% for the recommendation of reviewers (top-1 recommendations) and a *recall* of 71% for top-10 recommendations. As a complement of the previous study, Yu et al. [36] adopted Support Vector Machines (SVM) as the classification method and considered the files' location to recommend pull request reviewers. The experiments were carried out over 84 open-source projects and used a dataset containing 105,123 pull requests. The results presented *precision* of 67.3% and *recall* of 25.2% for top-1 recommendations, and *precision* of 33.8% and *recall* of 79.0% for top-10 recommendations.

Limeira et al. [11] employed classification algorithms for recommending reviewers to pull requests. The evaluation considered 48,510 pull requests, characterized by 14 attributes. These pull requests were obtained from 21 open-source projects, which were heterogeneous in terms of amount of pull requests. Which one of the experiments found *accuracy* ranging from 22.45% to 68.27% for recommending top-1 developers. The Random Forest classifier achieved the best result in 76% of the projects. In another experiment, when suggesting top-3 developers to analyze a pull request, the chance of identifying the developer that actually analyzed the pull request ranged from 47.33% to 95.47%.

Jiang et al. [12] proposed the CoreDevRec approach, which adopts an SVM classifier to recommend the five most adequate developers to review a pull request (top-5 recommendation). The approach adopted predictive attributes such as: social relationship, modified files' locations, and core team members' recent activities. The experiments were carried out over 5 projects, with a total of 18,651 pull requests. In the employed evaluation strategy, pull requests were organized in partitions, each one containing pull requests created in the same month. Thus, at every run, the classifier was trained with instances of a month and tested with instances of the following month, respecting the chronological order of creation of the contributions, that is, only previous information was used to conduct the recommendation. The approach's *accuracy* was calculated as the average of the *accuracy* obtained from each test run. The results revealed that CoreDevRec obtained *accuracy* ranging from 49.3% to 72.3% for top-1 recommendations, from 63.8% to 88.4% for top-2 recommendations, from 72.9% to 93.5% for top-3 recommendations, from 77.2%

to 95% for top-4 recommendations, and from 80.2% to 95.8% for top-5 recommendations.

Ying et al. [13] also investigated the recommendation of reviewers to pull requests. In the same fashion as in Yu et al. [36], the authors considered the similarity of title, description words, and comments made by collaborators when discussing about each pull request. They, however, employed the Random Walk algorithm [37] to compile a list of recommended reviewers. The approach was evaluated over 9 open-source projects with a total of 16,683 pull requests. The obtained results indicate that the proposed method outperforms the other methods in some cases. For example, the *precision* in project *akka* reaches 72% in average with 52% of *recall*, which improves the measure from 31% to 44% compared with another methods. The *precision* ranges from 26% to 72%, while the *recall* varies from 20% to 52%, considering the top-5 recommendations.

Rahman et al. [14] proposed an improvement to the technique file location introduced by Yu et al. [36] to leverage developers recommendation of reviewers to pull requests. The authors compared libraries of open pull requests to the ones of previously closed pull requests in order to obtain a similarity score between them, which then are assigned to each reviewer who has made comments on previous pull requests at least once. In their approach, the experience of a reviewer is determined by the sum of all similarity scores from pull requests on which he or she has made a comment. The evaluation of this approach was carried out over 10 commercial projects and 6 open-source projects, totaling 17,115 pull requests. The results indicated that the approach recommended code reviewers with *accuracy* of 85% to 92%, *precision* of 83% to 86%, and *recall* of 79% to 81%.

The recommendations provided by the aforementioned approaches mostly replicate historical patterns of software projects. They work as black boxes, not allowing a deeper analytical comprehension of the factors that motivate each specific recommendation. Uncovering the reasoning behind the recommendation is important to reveal possible conflicts of interest among requesters and reviewers, unbalanced workloads among reviewers, and regions of the codebase with few experts, representing a risk in face of turnover.

6.3 Materials and Methods

In order to answer the research questions laid out in Section 6.1, we adopted the extraction of association rules as indicate in Chapter 1. For some of the research questions, we performed additional analyses to assist in the comprehension of the obtained results. The paragraphs in the following describe the analyses that were made to answer each of the research questions.

RQ1 – Do the pull request characteristics influence on the reviewer assignment? In order to answer this question, we mined association rules that contain attributes such as the number of commits and the number of modified files in the pull requests. Apart from performing an evaluation of some association rules, we carried out a qualitative analysis to better comprehend the results. Section 6.4.1 presents the results of these analyses.

RQ2 - Does the requester profile influence on the reviewer assignment? We detected relationships between the lack of experience of the requester and the assignment of the reviewer. Complementarily, we explored rules considering the requester's affiliation with the project, that is, whether he or she is an external collaborator or a core team member. The results are discussed in Section 6.4.2.

RQ3 - Does the social relationship between requester and reviewer influence on the reviewer assignment? In order to answer this research question, we took into consideration rules with attributes that allow the detection of the social relationship, via GitHub, between requester and reviewer. First, we observed whether the reviewer follows the requester. We also did a deeper analysis by mining rules involving specific developers from each project. The results are discussed in Section 6.4.3.

RQ4 - Does the location of a pull request influence on the reviewer assignment? We mined rules with respect to the directories where the changed artifacts were stored. However, once the number of subdirectories is large, we preprocessed the data by using an attribute selection strategy based on the Information Gain measure [31], which evaluates the relevance of an attribute by measuring its information gain related to the target attribute. Subsequently, we selected the 10 most selective directories concerning the reviewer. Complementarily, we also investigated whether the reviewer is more likely to analyze pull requests that contain artifacts recently modified by him or her. Section 6.4.4 presents the results of these analyses.

We randomly selected three popular projects from 30 projects in GitHub, namely: Docker[1], Rails[2], and Kodi[3]. We assumed that the number of pull requests is a strong indication that the project adopts pull requests as a process for systematization of in-

[1]https://www.docker.com/
[2]http://rubyonrails.org/
[3]https://kodi.tv/

ternal/external contributions. We extracted 22,523 pull requests (excluding self-analyzed pull requests) out of these projects. Table 6.1 shows some characteristics of these projects: Number of pull requests, acceptance rate, number of distinct reviewers, number of distinct requesters, number of pull requests filed by core team members, and number of pull requests filed by external developers. Once the reviewer attribute has distinct values in each project, i.e., each project has different reviewers, we extracted the rules individually for each project. In this study, we used attributes of different dimensions (physical, social, location) as an attempt to understand the different factors that might influence on the reviewer assignment. In this study we mining rules with minimum support of 0.1%, given the large number of instances of the dataset and the reduced number of reviewers. In addition, we have defined a minimum confidence of 5%. We use these values to get as many rules as possible about the nature of the pull request paradigm. In this way, we were able to extract a large number of rules and ensure that they did not happen at random. As mentioned before, our studies we conducted experiments with most of the attributes extracted via GHTorrent. However, only a small set of them revealed rules with appropriate semantics in this scenario. The attributes discussed in the analysis are presented in Table 6.2. In order to transform numerical attributes into discrete ones, we use the discretization by simple binning, ignoring the class attribute to be defined, but considering frequency distribution. All labels have statistical representation in the database.

Table 6.1: Characteristics of the analyzed projects

Caracteristics	**Projects**		
	Docker	**Rails**	**Kodi**
# Pull Requests	6,979	11,728	3,816
Acceptance Rate (%)	84.1	75.0	84.0
# Different Reviewers	45	44	61
# Different Requesters	1,437	2,934	560
# PRs submitted by core team members	248	569	70
# PRs submitted by external members	6,731	11,159	3,748

6.4 Results and Discussion

In this Section, we discuss the results of our analyses and present answers to the research questions defined in Section 6.3. Section 6.4.1 presents a discussion on pull request characteristics that influence on the reviewers assignment. Section 6.4.2 brings the discussion on factors related to the requester profile influencing the reviewers assignment. Section 6.4.3

Table 6.2: Attributes used in this study.

Attribute	Description
commits_pull	the number of commits in the pull request, with values: "1 commit"; "some commits": 2 to 4 commits; and "many commits": more than 4 commits.
developer_type	whether the requester is internal or external.
directory_names	represents a set of 10 attributes, where each is the name of a directory. The value "true" indicates that a file from that directory was changed, the value "false" indicates otherwise.
files_changed	the number of added, removed, or edited files, with values: "1 file"; "some files": 2 to 4 files; and "many files": more than 4 files.
first_pull	whether the pull request is the first performed by the requester.
recent_committer	the name of the developer responsible for the larger number of commits in the files changed by the pull request in the last seven days.
requester_pull	the name of the developer that filed the pull request.
reviewer_follows_requester	whether the reviewer follows the requester on GitHub.
reviewer_pull	the name of the developer responsible for reviewing the pull request.
status_pull	the final status of the pull request: "accepted" (merged) or "rejected" (closed).

presents the influence of social factors on the assignment of reviewers for pull requests. Then, Section 6.4.4 brings the results on how the reviewers assignment can be influenced by the location of the changes in a pull request.

6.4.1 RQ 6.1 – Do the pull request characteristics influence on the reviewer assignment?

Some characteristics of the pull requests, such as the number of changed files and the number of commits, are known by the time of its submission. Although these characteristics can change during the review process, they may indicate, someway, how complex the review process will be since the pull request submission.

Our results show that the number of commits associated with pull requests may influence the reviewer assignment. Table 6.3 displays association rules (in decreasing order of *Lift* values) of the type *commits_pull = "1 commit" → reviewer_pull* extracted from projects Docker, Rails, and Kodi. Rule 1 indicates that the reviewer's chance of being "runcom" increases in 22% when a pull request has only "1 commit". The rules evidence that, in all of the investigated projects, certain reviewers have higher chances of analyzing pull requests with only one commit, that is, theoretically less complex. Conversely, this does not necessarily mean that, among all the reviewers, only they analyze simpler contributions, but the chances are higher for those cases. When we analyze the *confidence* values of the reverse of each of the rules ($Y \rightarrow X$), we note that they are significantly larger than the ones of type $X \rightarrow Y$. Thus, we conclude that some reviewers opt to review pull requests with a single commit. For instance, Rule 7 implies that 100% of the pull requests reviewed by *dmathieu* possess a sole commit. Similar knowledge can be inferred from the analysis of the remainder of the rules.

Table 6.3: Association rules of type *commits_pull = "1 commit" → reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	***Sup* (%)**	***Conf* X → Y (%)**	***Conf* Y → X (%)**	***Lift***
1	Docker	*"runcom"*	1.0	0.8	97.2	1.22
2	Docker	*"ostezer'*	1.0	0.9	96.3	1.21
3	Kodi	*"ace20022"*	0.4	0.5	82.4	1.21
4	Kodi	*"Jalle19"*	0.5	0.7	86.4	1.19
5	Docker	*"thaJeztah"*	2.0	2.8	93.5	1.18
6	Kodi	*"mkortstieg"*	3.0	5.1	84.8	1.17
7	Rails	*"dmathieu"*	0.1	0.1	100.0	1.15
8	Rails	*"seuros"*	1.0	1.3	98.5	1.14
9	Rails	*"schneems"*	0.5	0.6	98.4	1.14

When we analyze rules in which *commits_pull = "many commits"* is the antecedent, we also verify that there is a correlation with the reviewers. Rules in Table 6.4 illustrate this pattern. In projects Rails, Docker, and Kodi, when a pull request has more than 4 commits, the chances of being reviewed by certain reviewers increase, with a variation in the strength of the influence in each case. For example, the *Lift=2.91* indicates that reviewer's chance of being *"shykes"* increases in 191% when a pull request has many commits. Unlike the previous conclusion, the analysis of the *confidence* values does not allow a generalization of the influence of X on Y or vice-versa. The differences between the *confidence* values are not as significant, with the exception of Rules 6, 7, 8, and 9, in which a large amount of commits seems to influence the reviewer attribute.

Despite the existence of a good practice in the Rails project suggesting the *"squashing of commits"*[4], pull requests with several commits are also submitted – approximately 1.85% of the total. With that said, in order to identify potential influence factors on the reviewers assignment for more complex contributions, we carried out a qualitative analysis.

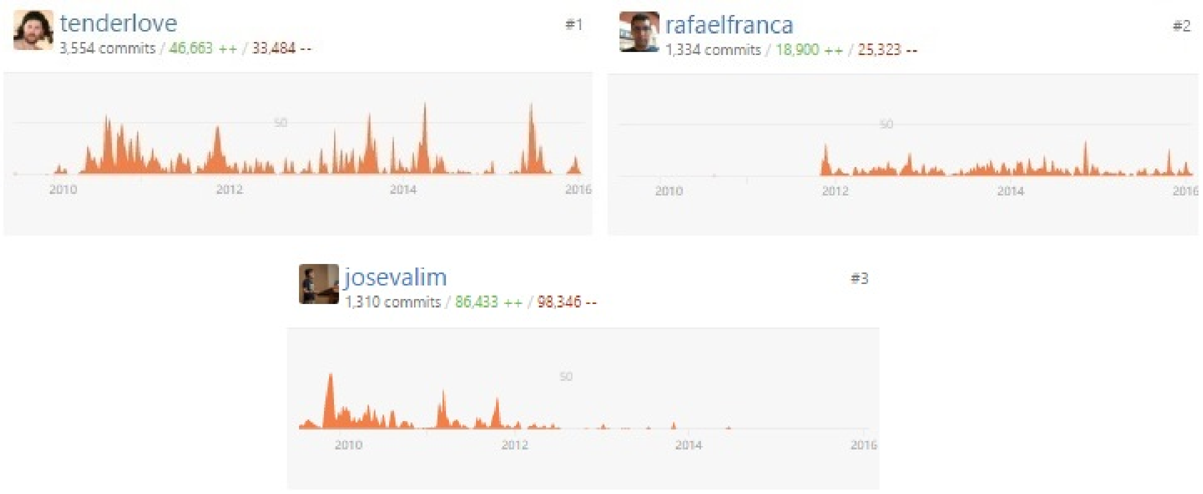
[4]http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html#iterate-as-necessary

Table 6.4: Association rules of type *commits_pull = "many commits"* → *reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	***Sup*** (%)	***Conf*** **X → Y** (%)	***Conf*** **Y → X** (%)	***Lift***
1	Docker	*"shykes"*	0.6	14.4	12.2	2.91
2	Docker	*"creack"*	0.6	13.7	9.2	2.20
3	Docker	*"unclejack"*	0.2	5.8	7.4	1.78
4	Kodi	*"davilla"*	0.4	5.1	10.2	1.43
5	Rails	*"tenderlove"*	0.1	7.4	2.5	1.36
6	Kodi	*"jmarshallnz"*	2.0	27.1	9.5	1.32
7	Rails	*"rafaelfranca"*	0.6	34.3	2.2	1.17
8	Kodi	*"Martijnkaijser"*	2.0	28.6	8.3	1.16
9	Rail	*"josevalim"*	0.2	13.0	2.1	1.12

We assumed the hypothesis that developers with greater activity[5] in the repository could review more complex contributions, given their experience and knowledge in the project. Rules 5, 7, and 9 in Table 6.4 showed that reviewers *"tenderlove"*, *"rafaelfranca"*, and *"josevalim"* are the ones with the highest chances of analyzing pull requests with several commits in project Rails. Our qualitative analysis evidenced that these reviewers are the three developers with the most commits in project Rails during the aforementioned period, as shown in Figure 6.1. This analysis provides initial evidences to our hypothesis. We deem this result very significant, since project Rails has 3,184 developers who had already committed to the repository.

Figure 6.1: Top-3 developers with more commits in Rails project.


tenderlove
3,554 commits / 46,663 ++ / 33,484 --
#1
50
2010
2012
2014
2016
rafaelfranca
1,334 commits / 18,900 ++ / 25,323 --
#2
50
2010
2012
2014
2016
josevalim
1,310 commits / 86,433 ++ / 98,346 --
#3
50
2010
2012
2014
2016

[5]We measured activity by the number of commits filed by developers throughout their time of participation in the project until the date in which the data was collected – 09/03/2015 for Docker, 09/26/2015 for Kodi, and 09/02/2015 for Rails

We also performed the same qualitative analysis on projects Docker and Kodi, in order to explain why some reviewers have higher chances of analyzing pull requests with many commits. On Docker, the most likely pull request reviewers are *"shykes"*, *"creack"*, and *"unclejack"*. These three occupy, respectively, positions 1, 3, and 12 in the rank of developers with the highest amount of commits, within a total of 1,572 committers. This result is also deemed rather relevant, if we take into account that these three reviewers are among the 1% developers who have performed the most commits to the repository. In project Kodi, the reviewers *"davilla"* and *"Martijnkaijser"* occupy positions 3 and 12, respectively, in the rank of committers, from a total of 503 developers; that is, they are among the 3% most active developers in terms of number of commits. We could not determine the position of reviewer *"jmarshallnz"*; there may have been a username change, arguably. All in all, the qualitative analysis carried out on projects Docker and Kodi also confirmed that experience and knowledge in the project may be factors of influence on the assignment of developers to review more complex pull requests.

In a similar fashion to the number of commits, the amount of files changed by a pull request is a characteristic that may indicate how complex its review process will be, thus becoming an influence factor on the reviewer assignment. We extracted rules of type *files_changed = "1 file" → reviewer_pull* from project Rails, Docker, and Kodi, which are showcased in Table 6.5 (rules are ordered by *Lift* values in decreasing order).

Table 6.5: Association rules of type *files_changed = "1 file" → reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	**Sup (%)**	***Conf* X → Y (%)**	***Conf* Y → X (%)**	***Lift***
1	Rails	*"claudiob"*	0.3	0.5	91.4	1.88
2	Rails	*"zzak"*	0.2	3.8	88.5	1.82
3	Rails	*"seuros"*	0.1	2.0	87.7	1.81
4	Kodi	*"alanwww1"*	1.0	2.4	81.8	1.64
5	Docker	*"mzdaniel"*	0.4	0.8	82.4	1.62
6	Docker	*"ostezer"*	0.6	1.2	77.8	1.53
7	Docker	*"jamtur01"*	4.0	8.4	74.5	1.46
8	Kodi	*"Jalle19"*	0.4	0.8	72.7	1.46
9	Kodi	*"mkortstiege"*	2.0	5.5	63.2	1.27

Results suggest that some developers have higher chances to analyze pull requests with a single modified file. On project Rails, for instance, the chances of contributions displaying such characteristic being analyzed by reviewers *"caludiob"*, *"zzak"*, and *"seuros"* are raised in 88% (*Lift=1.88*), 82% (*Lift=1.82*), and 81% (*Lift=1.81*), respectively.

A similar pattern can be inferred in projects Docker and Kodi. Rules in which the occurrence of the antecedent *files_changed = "1 file"* increases the chances of assigning certain reviewers (Rules 4 through 9) were extracted from both projects.

Assuming that having just one changed file may imply low complexity in the review of the pull requests, we also performed a rule *confidence* analysis for this scenario. We can observe in Table 6.5 that, in every case, the rule *confidence* in the direction $Y \rightarrow X$ is significantly larger than in $X \rightarrow Y$, evidencing that some reviewers opt to review less complex contributions in terms of number of changed files.

We also extracted rules of the type *files_changed = "many files" → reviewer_pull*, shown in Table 6.6. We aim at verifying whether pull requests having more than four modified files are assigned to specific developers. In this analysis, we also uncovered rules with elevated *Lift* values, implying that some reviewers have greater chances of analyzing pull requests with several files changed – Rules 1 through 9 evidence that.

Table 6.6: Association rules of type *files_changed = "many files" → reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	***Sup* (%)**	***Conf* X → Y (%)**	***Conf* Y → X (%)**	***Lift***
1	Rails	*"dhh"*	0.2	2.2	27.1	2.43
2	Kodi	*"xhaggi"*	0.4	1.9	42.9	2.06
3	Docker	*"icecrime"*	1.0	6.9	34.8	1.79
4	Rails	*"jeremy"*	0.4	4.4	18.3	1.64
5	Rails	*"sgrif"*	0.3	3.4	18.1	1.62
6	Docker	*"tiborvass"*	1.0	7.1	28.0	1.44
7	Kodi	*"davilla"*	1.0	4.8	27.7	1.34
8	Kodi	*"wsoltys"*	0.3	1.5	26.7	1.28
9	Docker	*"shykes"*	1.0	6.2	24.5	1.25

6.4.2 RQ 6.2 - Does the requester profile influence on the reviewer assignment?

We mined rules involving characteristics from the requester profile herein. Then, we analyzed the influence of those characteristics on the reviewer assignment. The first factor that we tackled was the requesters' lack of experience with pull-based development. We consider to be inexperienced a developer who has not contributed via pull requests previously in a given project, that is, we extracted rules in which the attribute *first_pull = true* is the antecedent and *reviewer_pull* is the consequent. Table 6.7 dis-

RQ 6.1 - Do the pull request characteristics influence on the reviewer assignment?
Answer: Some reviewers have preference for pull requests with few commits and few changed files, while others tend to analyze more complex contributions.
Relevance: Lima Júnior et al. [11] and Jiang et al. [12] employed some pull request characteristics as attributes to recommend reviewers. However, as any black-box approach, they did not indicate the extent of the influence of these attributes on the assignment of reviewers. Moreover, they did not qualitatively investigate such influence. Yu et al. [36, 10], Jiang et al [13], and Rahman et al. [14] did not consider pull request characteristics as predictive attributes.
Implications: Our results indicate that an unbalancing workload among reviewers is likely occurring in some projects, as some reviewers always handle simple pull requests while others frequently handle more complex ones. These results may also indicate unbalanced skill distribution among developers, as just few reviewers handle most of the difficult pull requests. Strictly following recommendations provided by machine learning tools would possibly reinforce such undesired scenarios.

plays the results of this analysis. The rules allow the conclusion that some reviewers tend to evaluate pull requests submitted by inexperienced developers. Rules 1 and 2, for instance, show that when *first_pull = true* for a requester in Docker project, reviewers *"kencochrane"* and *"mzdaniel"* have the chances of reviewing such contributions increased in 155% (*Lift = 2.55*) and 145% (*Lift = 2.45*), respectively. Even though the table is sorted by *Lift* in decreasing order, it is noticeable that this pattern is also present in the other projects.

Table 6.7: Association rules of type *first_pull = "true"* $\rightarrow$ *reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	***Sup* (%)**	***Conf* X → Y**	***Conf* Y → X**	***Lift***
1	Docker	*"kencochrane"*	0.3	1.1	58.1	2.55
2	Docker	*"mzdaniel"*	0.3	1.2	56.0	2.45
3	Rails	*"laurocaetano"*	0.1	0.4	55.6	1.86
4	Docker	*"jamtur01"*	2.4	10.6	42.2	1.85
5	Rails	*"arunagw"*	0.5	1.7	53.2	1.79
6	Kodi	*"alanwww1"*	0.4	2.5	25.5	1.71
7	Kodi	*"Memphiz"*	0.8	5.3	23.3	1.56
8	Rails	*"steveklabnik"*	0.8	2.6	45.6	1.53
9	Kodi	*"Montellese"*	0.9	5.8	22.9	1.52

We additionally carried out a *confidence* analysis to evaluate the direction of the correlation between *first_pull = "true"* and *reviewer_pull.* In some of the cases shown on Table 6.7, the *confidence* values of rules of the type $Y \rightarrow X$ are significantly superior to those of the type $X \rightarrow Y$. Therefore, we can conclude that some reviewers have a preference for the review of contributions made by inexperienced requesters, as observable in Rules

1, 2, 3, and 5.

We also extracted rules of the type *first_pull = "false"* $\rightarrow$ *reviewer_pull* with the intent of verifying whether certain reviewers have preference for analyzing pull requests from requesters who have previously contributed. Figure 6.2 shows the *Lift* values for the rules extracted from our project corpus, in which the identifiers (#1, #2, and #3) represent the rules having the highest values of *Lift* for each project. It is noticeable that despite the existence of rules with *Lift* greater than 1, the chances of *reviewer_pull* occurring does not significantly increase when *first_pull = "false"* occurs. As a result, we can draw the conclusion that requesters who exhibit previous experience with the paradigm do not influence much on the reviewers assignment.

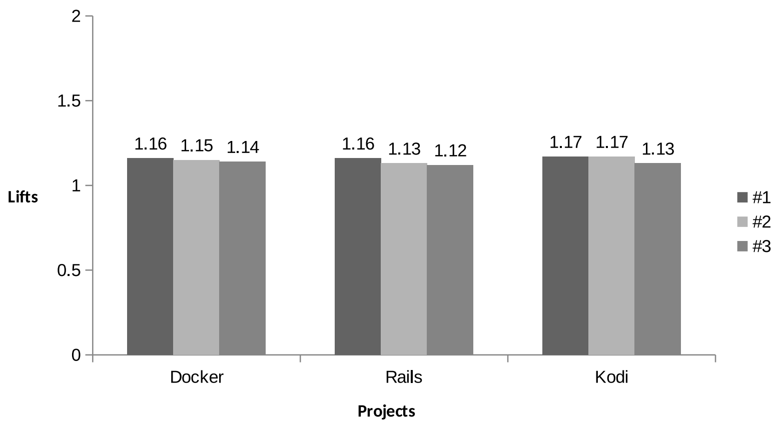


Figure 6.2: *Lifts* of the rules of type: *first_pull = "false"* $\rightarrow$ *reviewer_pull.*

Another feature related to the requester's profile that we investigate herein is whether he or she is an external developer or a core team member. In general, pull requests are submitted by developers who are not part of the core team (i.e., external), but who still desire to contribute somehow. Nonetheless, this mechanism may also aid in the systematization and integration of code implemented by core team members, so that other members are able to peer review their contributions prior to the integration in the repository. In the light of that, we conducted the extraction of association rules of the form *developer_type* $\rightarrow$ *reviewer_pull* from projects Docker, Rails, and Kodi. We first extracted rules wherein *developer_type = core team* is the antecedent. Table 6.8 shows the main extracted rules, sorted by *Lift*. The results indicate that, in some projects, there are reviewers who have higher chances of evaluating pull requests submitted by members of the core team. In Docker, for instance, the chances of reviewer *"metalivedec"* being assigned to pull requests filed by core team members is increased in 71% (*Lift=1.71*), whereas in Rails the most likely reviewer of pull requests submitted by core team members is *"tenderlove"*, with chances increased in 64% *(Lift=1.64)*. In this analysis, we could not extract rules from project Kodi, considering the minimum *support* of 0.3%.

Table 6.8: Association rules of type *developer_type = "core team"* → *reviewer_pull* in Docker and Rails.

#	**Project**	**Consequent**	***Sup*** (%)	***Conf*** X → Y (%)	***Conf*** Y → X (%)	***Lift***
1	Docker	*"metalivedec"*	0.3	7.3	6.1	1.71
2	Docker	*"SvenDowideit"*	0.3	8.1	6.0	1.69
3	Rails	*"tenderlove"*	0.4	9.0	8.0	1.64
4	Rails	*"chancancore"*	0.1	2.1	7.9	1.63
5	Rails	*"jonleigthon"*	0.1	2.6	6.4	1.33
6	Docker	*"shykes"*	0.2	6.0	4.3	1.22

Despite knowing that external members submit the majority of pull requests, we mined rules with the intent of verifying whether there exist developers with higher chances of reviewing contributions made by this type of requesters. Figure 6.3 displays *Lift* values for rules of the form *developer_type = "external"* → *reviewer_pull* in projects Docker, Rails, and Kodi. The identifiers (#1, #2, and #3) indicate the rules with the highest value of *Lift* within the projects. Unlike the findings when the requester is a core team member, if he or she is external to the project, the chances of having specific reviewers assigned to his or her pull request do not increase significantly.

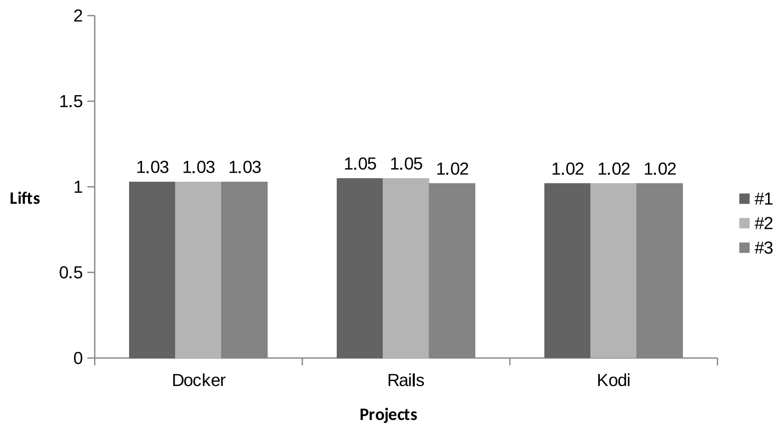


Figure 6.3: *Lifts* of the rules of type: *developer_type = "external"* → *reviewer_pull.*

RQ 6.2 - Does the requester profile influence on the reviewer assignment?
Answer: Some reviewers frequently (up to 58% of the times) analyze pull requests filed by inexperienced requesters. Other reviewers have more chances (up to 71% of the times) to analyze pull requests filed by core team members.
Relevance: As far as we know, our study is the first to bring forth results that indicate the influence of requester profile characteristics on the reviewer assignment. Previous work did not considered the requester inexperience and developer type as possible reasons to influence assignment of reviewers.
Implications: Our results indicate that some reviewers are in charge of assessing the contributions of newcomers. This task is fundamental in open-source projects, as reception issues [38] are key barriers for onboarding newcomers in a project. Moreover, other reviewers focus on analyzing internal pull requests, filed by core team members. These reviewers are not gatekeepers, as all core team members already have *push* privileges to the repository. They act as quality assurance for the project. In both cases, managers should be aware of who is performing each task, as inappropriate assignments may frighten newcomers or jeopardize the harmony of the core team.

6.4.3 RQ 6.3 - Does the social relationship between requester and reviewer influence on the reviewer assignment?

Repository hosting tools such as GitHub provides social networking resources, allowing developers to have followers and social relationships among them. Our study also investigated the impact of some social factors on the reviewer assignment to analyze pull requests.

We investigated the relationship between the reviewer and the requester as a first social factor. We extracted rules of the form *reviewer_follow_requester = "true"* $\rightarrow$ *reviewer_pull*, that is, rules that indicate the increase of the chances of a given reviewer assignment when he or she follows the requester on GitHub. Table 6.9 shows the rules and their interest measures on projects Docker, Rails, and Kodi. The results suggest that such pattern not only exists, but is also very prominent. This table illustrates only the three most significant rules for each project – with the exception of Docker, which has only two rules whose *Lift* values surpassed the 1.0 threshold. For example, Rule 1 shows that in project Kodi, when *"fritsch"* follows the requester, the chances of having him as reviewer increases 26 times *(Lift=26.00)*. Rules 2 through 8 imply the same pattern, but with smaller values of *Lift*.

Based on the hypothesis that social relationships may influence on the reviewer assignment, we seek to determine whether pull requests submitted by a given requester has higher chances of being analyzed by specific reviewers. For that, we explore the social

Table 6.9: Association rules of type *reviewer_follow_requester = "true" → reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Consequent**	***Sup*** (%)	***Conf*** **X → Y** (%)	***Lift***
1	Kodi	*"fritsch"*	0.8	41.6	26.00
2	Docker	*"unclejack"*	1.8	35.1	10.73
3	Rails	*"guilleiguaran"*	0.8	31.7	10.48
4	Kodi	*"opdenkamp"*	0.47	23.4	6.19
5	Rails	*"arthurnn"*	0.1	5.4	4.84
6	Rails	*"robin850"*	0.1	3.8	4.10
7	Docker	*"vieux"*	1.7	32.6	4.05
8	Kodi	*"LK4D4"*	0.8	7.9	2.36

relationship factor more closely, with a finer granularity in the pull request context. To that effect, we mined rules of the form *requester_pull → reviewer_pull* from projects Docker, Rails, and Kodi.

Table 6.10 shows the mined rules and their interest measurements wherein the name of *requester_pull* is the antecedent and *reviewer_pull* is the consequent. Results suggest that the requester imposes a strong influence on the reviewer assignment. For instance, Rule 1 indicates that, on project Kodi, when *"CutSickAss"* sends a pull request, the chances of it being reviewed by *"alanwww1"* increase by more than 60 times *(Lift=60.13)*. Besides, even though *"alanwww1"* reviews pull requests by several requesters, he is responsible for reviewing 86.7% of the pull requests filed by *"CutSickAss"*.

Table 6.10: Association rules of type *requester_pull → reviewer_pull* in Docker, Rails, and Kodi projects.

#	**Project**	**Antecedent**	**Consequent**	***Sup*** (%)	***Conf*** **X→Y** (%)	***Lift***
1	Kodi	*"CutSickAss"*	*"alanwww1"*	0.3	86.7	60.13
2	Rails	*"tomkadwill"*	*"zzak"*	0.1	61.9	29.75
3	Rails	*"roankjangir47"*	*"kaspth"*	0.1	30.0	25.45
4	Kodi	*"fetzerch"*	*"opdenkamp"*	0.4	75.0	19.87
5	Kodi	*"phil65"*	*"ronie"*	0.3	38.9	19.52
6	Rails	*"robertomiranda"*	*"guilleiguaran"*	0.2	55.1	18.20
7	Docker	*"moxiegirl"*	*"thaJeztah"*	0.2	39.5	16.50
8	Docker	*"SvenDowideit"*	*"fredlf"*	1.0	22.9	12.36
9	Docker	*"mzdaniel"*	*"shykes"*	0.2	50.0	10.11

Similar patterns to the one revealed by Rule 1 are present in all three projects, as observed in the remaining rules. When we analyze the three most significant rules from each project, the lowest *Lift* values found were 10.11 (Rule 9), 18.20 (Rule 6), and 19.52 (Rule 5) on projects Docker, Rails, and Kodi, respectively. This result becomes even more relevant if we take into consideration the large number of rules extracted from projects with significant positive *Lift* values. For instance, from project Docker, we mined 78 rules of the form *requester_pull* → *reviewer_pull* with *Lift* greater than or equal to 1.10. Similarly, the number of rules with *Lift* values greater than or equal to 1.10 was 101 in project Rails and 54 in Kodi.

RQ 6.3 - Does the social relationship between requester and reviewer influence on the reviewer assignment?
Answer: The chances of having some reviewers analyzing pull requests increases (in up to 26 times) when the contributions are filed by requesters of their acquaintance. Other reviewers have their chances to analyze pull requests increased (in up to 60 times) when pull requests are filed by specific requesters.
Relevance: Jiang et al. [12] was the only author to adopt the correlation of personal relationship and reviewer assignment for recommending reviewer. However, our results allow a quantification of the extent of such pattern by means of metrics such as *support*, *confidence*, and *Lift*. The studies in [10, 11, 13, 14, 36] did not consider these factors as possible reasons to influence assignment of reviewers.
Implications: Although in some cases requesters and reviewers may have similar technical background, in other cases the social factor may indicate a conflict of interest. Managers could use this insight to alternate the paring of requesters and reviewers from time to time to mitigate conflicts of interest and disseminate knowledge.

6.4.4 RQ 6.4 - Does the location of a pull request influence on the reviewer assignment?

As previously mentioned, pull requests are employed in the context of collaborative software development using version control systems. Even though several developers may be able to change different files in a project, a number of them work more on some files than others, generally. As a result, an ownership profile takes shape over such files. Furthermore, directories behave in a similar way, as software repositories allow the storage of files organized according to a hierarchical directory structure. Consequently, the directory wherein a file is stored determines its location. In this sense, we mined rules taking into account the influence of location in terms of directories.

We first extracted rules of the form *recent_committer* → *reviewer_pull*, wherein the antecedent's and the consequent's labels are the same. With this analysis we aim at verifying whether having a developer committed recently on the files of directories manipulated in a pull request increases the chances of he or she becoming the reviewer of such contribution. Figure 6.4 showcases the *Lift* values of the ten most significant rules of this form in project Docker. Results confirm the existence of this pattern, that is, when a developer was the most active, in terms of commits, on the changed files present in directories in a pull request during the previous week, he or she has greater chances to become the reviewer. In project Docker, for example, the chances of having *"moxiegirl"* as reviewer of pull requests increases in 8 times *(Lift=9.03)* when some files modified in the are within directories contributions have been modified by her recently. This pattern is observable, with varying degrees of intensity, to other developers of project Docker.

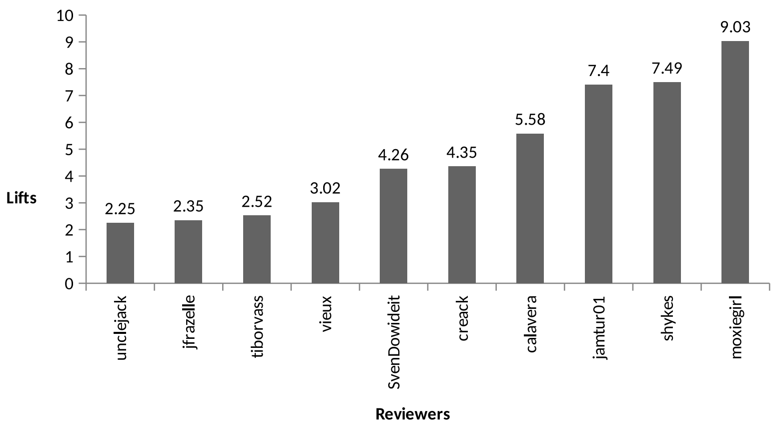


Figure 6.4: *Lifts* of the rules of type: *recent_commiter* $\rightarrow$ *reviewer_pull* in Docker project.

Aiming at confirming the existence of this pattern in the other projects, we repeated the experiment and attested its existence. Figures 6.5 and 6.6 show *Lift* values for rules extracted from projects Rails and Kodi, respectively. Rules with even stronger *Lift* were encountered in both projects. For instance, in Rails, the rule *recent_committer = "jonleighton"* $\rightarrow$ *reviewer_pull = "jonleighton"* has a *Lift* value of 14.4. As to Kodi, the strongest rule is that which involves developer *huceke*, with a *Lift* value of 21.33.

We also investigated whether the location of the pull request also affects the reviewer assignment. As previously mentioned, we consider the location of the pull request as the paths of directories within the repository wherein modified files are stored. Therefore, we perform the rules extraction in project Rails. By extracting rules with the attribute *location* we found that this factor exerts influence on *reviewer_pull*. Table 6.11 shows rules of the form *directory_names* $\rightarrow$ *reviewer_pull* and unravels the knowledge extracted based on the correlation of these factors.

Though pull request location is not voluntarily determined by the requester, but by the need of a contribution in itself, the results allow for the conclusion that, depending

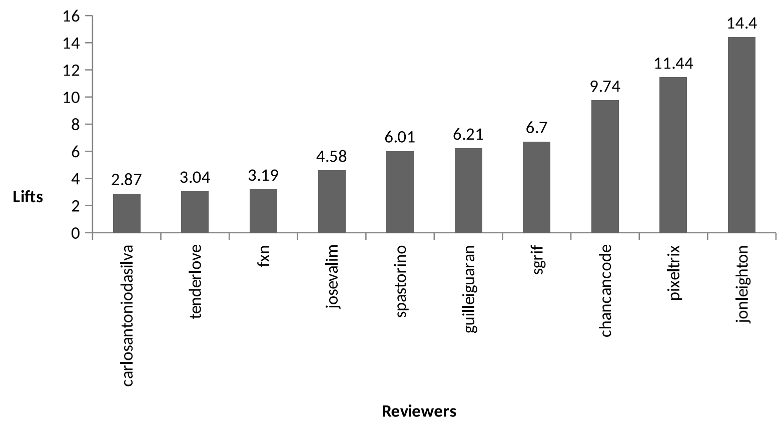


Figure 6.5: *Lifts* of the rules of type: *recent_commiter* → *reviewer_pull* in Rails project.

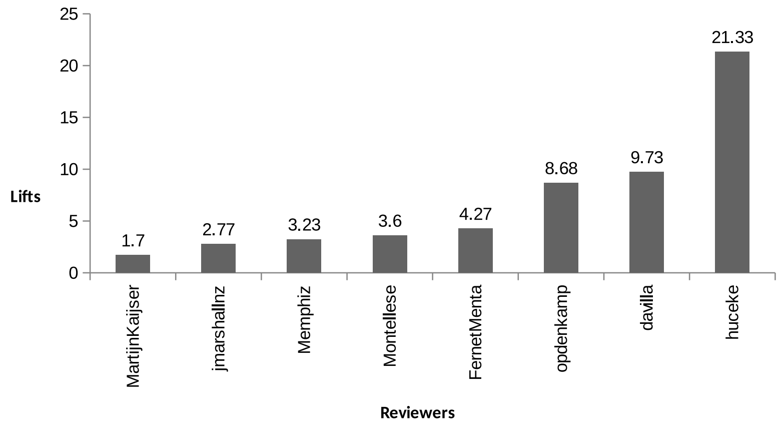


Figure 6.6: *Lifts* of the rules of type: *recent_commiter* → *reviewer_pull* in Kodi project.

on the location of the change, the contributions may be led to analysis by specific reviewers, as seen in Table 6.11. For instance, pull requests which manipulate files located in the directory *"...guides/source"* are more likely to be analyzed by reviewers *"zzak"* and *"senny"*, according to the observations made from Rules 1 and 2, respectively. Pull requests at this location are 346% *(Lift=4.46)* more likely to be analyzed by *"zzak"*. This location increases the chances of *senny* in 83% *(lift=1.83)*. Rules 3, 4, 5, and 6 evidence other cases in which locality influences the reviewer on project Rails.

6.5 Threats to Validity

Although we have taken care to reduce the threats to validity of our study, a few uncontrolled factors may have influenced the observed results. Regarding internal validity,

Table 6.11: Association rules of type *directory_names* → *reviewer_pull* in Rails project.

#	**Antecedent**	**Consequent**	***Sup*** (%)	***Conf*** **X→Y** (%)	***Lift***
1	*directory_names = "... guides/source"*	*"zzak"*	1.1	9.3	4.46
2	*directory_names = "... guides/source"*	*"senny"*	1.7	13.9	1.83
3	*directory_names = "... activerecord/ test/cases"*	*"tenderlove"*	1.2	9.8	1.79
4	*directory_names = "... activerecord"*	*"senny"*	1.1	12.7	1.68
5	*directory_names = "... activerecord"*	*"rafaelfranca"*	3.5	41.5	1.42
6	*directory_names = "... activerecord/test/cases"*	*"rafaelfranca"*	4.3	35.3	1.21

RQ 6.4 - Does the location of a pull request influence on the reviewer assignment?
Answer: The chances of having some reviewers analyzing pull requests increases (in up to 21 times) when the pull requests change files in directories that the reviewers have recently changed. Other reviewers have their chances to analyze pull requests increased (almost to 4 times) when pull requests change files from specific directories.
Relevance: Yu et al. [36], Ying et al. [13], Rahman et al. [14] and Jiang et al. [12] take into account the locality of artifacts and state that it is a prominent predictor for reviewer. Notwithstanding, our results quantify the relevance of previous knowledge (in terms of recent commits or location) in the assignment of reviewers to pull requests.
Implications: Our results point out regions of the codebase with few experts. This represents a major risk in face of turnover, as few other developers would be able to assimilate the workload of a leaving developer.

our corpus, comprised of three projects, led to some association rules with low *support* values, but with a relevant number of rules. However, we discarded the possibility that these patterns might have arisen accidentally, since we identified a set of patterns whose frequent occurrence demonstrates the unlikeliness of appearing by chance. Even so, we acknowledge the need of corroboration of all obtained results with a study involving a much larger number of projects.

Finally, regarding external validity, our capability to generalize observations made throughout this study is restricted to projects with similar characteristics: all projects are open-source and have a certain degree of maturity regarding the use of pull requests. Therefore, we cannot generalize our results to industrial projects or even open-source ones with diverging characteristics. Future studies must be carried out over projects in different characteristics in order to deal with this threat.

6.6 Final Remarks

Herein we presented a study on the factors that influence the developer assignment to review pull requests in open-source projects. The knowledge attained by our results reveals patterns that may provide insights both to recommendation tool developers and project managers. These results may explain influence factors in unbalanced workloads among different reviewers. Moreover, the results may indicate regions of the codebase with few experts, representing a risk in face of turnover.

In particular, the results allow the following conclusions: (i) factors such as number of commits and files in the pull request may influence the reviewer assignment; for instance, some reviewers have preference for pull requests with few commits and files, in some cases with *confidence* ranging from 82% to 100%; (ii) factors regarding the requester's profile may influence on reviewer assignment; for instance, some reviewers frequently (up to 58% of the times) analyze pull requests filed by inexperienced requesters. Other reviewers even have their chances increased in 71% to evaluating pull requests filed by other core team members; (iii) the social relationship between requester and reviewer exerts influence on pull request review process, that is, when the reviewer knows the requester, his or her chances of evaluating such contributions may increase 26 times, in some cases. In addition, it indicates that depending on who submits the pull request, the chances of having some reviewer analyzing it increases (up to 60 times); and (iv) factors such as ownership and locality of pull request are important predictors for the reviewer, that is, we found cases where pull requests that changed files in directories that have been recently modified by a given reviewer increased his chances of evaluating this pull requests in 21 times. In addition, we also noticed that the location of files changed by the pull request can increase (in almost 4 times) the chances of it being analyzed by specific reviewers.

Chapter 7

Conclusion

This thesis presented a series of studies about the nature of the pull-based development paradigm. After collecting data from software repositories, we extracted patterns to aid in the characterization of the collaboration via pull requests. This characterization focused on four different perspectives: acceptance, rejection, lifetime, and reviewers assignment. Previous studies have been proposed in this direction, however they used black-box approaches and have neglected to explore some important and useful aspects, namely: the identification of characteristics with different dimensions (physical, social and metadata) which are influent in an isolated or conjoint fashion; the extent and strength of the influence of such characteristics; and a qualitative analysis to explain the occurrence of certain patterns.

7.1 Main Contributions

Our studies are based on an extensive analysis of pull requests, and the obtained results allow a clear understanding of their nature. As a component of this work, we identified the most relevant related work and executed several experiments to discover and analyze intrinsic matters regarding pull requests. In this sense, the main contributions of this thesis are:

1) **The identification of factors that influence on the acceptance of pull requests in open-source projects** – We explored 61,592 pull requests from 72 projects and identified factors that increase the chances of acceptance. Being aware of such factors is important to requesters and reviewers. The reviewers might identify trends in the collaborating community and establish contribution policies to prioritize the evaluation of pull requests with higher chances of acceptance. As for requesters, they may be able to

direct their actions in order to maximize the chances of acceptance of their contributions. Moreover, it allows the requesters to be more productive and efficient, since they attain higher chances of having their work accepted. Our main findings are:

a) Pull requests coded in certain programming languages, tend to have increased chances of acceptance such as Ruby (in up to 11%), C# (in up to 26%), Scala (in up to 35%), and Go (in up to 51%);

b) Physical features, such as the amount of commits present in the pull request, may influence on the acceptance. Pull requests with just one commit have the chances of acceptance increased in 7%;

c) The requester's lack of experience reduces the chances of acceptance in up to 32%, whereas pull requests submitted by core team members have 35% more chances of acceptance.

2) **The identification of factors that influence on the rejection of pull requests submitted by core team members in open-source projects with high acceptance rates** -– We extracted association rules from 20,140 pull requests of 7 software projects featuring high acceptance rates, taking into account only contributions that had been submitted by core team members. The features that influence the rejection reveal patterns to be avoided by core team members and, as a result, improve the quality of internal contribution activities. Our main findings are:

a) There is a correlation between the rejection of pull requests and their physical features. A large number of commits in a pull request raises the chances of rejection in 51%, while a large number of files raises in 44%;

b) The lack of experience of a core team member who submits a pull request increase the chances of rejection in 238%. Besides, pull requests having comments during their evaluation process increase the chances of rejection in 13%;

c) The location of the modified artifacts influences on the rejection of pull requests. In some cases, this may represent in up to 261% more chances of rejection. Moreover, the project contribution policies also affect the chances of rejection;

d) We also observed a joint influence of such factors on the chances of rejection. Patterns which increase in up to 623% the chances of rejection were evidenced when a pull request is the first submitted by a developer, has a large number of commits, and modifies a sensitive part of the code.

3) **The identification of factors that influence on the lifetime of pull requests in open-source projects** -- We collected 97,463 pull requests from 30 software projects and discovered patterns that indicate what influences on the pull request lifetime in terms of socio-technical aspects, external contributors, and the contribution process. Comprehending what influences the pull request lifetime is important because extended delays in the analysis of pull requests with significant chances of being accepted are not desired, since such delays might affect the involvement of the requester in the collaboration process. Moreover, these delays are negative for the introduction of bug fixes and features to the software product. Furthermore, our findings disclose to project managers the need for an adequate assignment of pull request to reviewers in order to decrease their lifetime. This may increase the team efficiency, once the time saved can be used to complete other tasks. Our main findings are:

a) There is a correlation between pull request lifetime and its acceptance/rejection, evidencing that contributions that have been quickly analyzed tend to be integrated (the chances in up to 11% increase), and those whose lifetime is long tend to be rejected (the chances in up to 114% increase). Moreover, the results provide preliminary evidence on the direction of this relation, indicating that lifetime implies acceptance rather instead of the opposite;

b) Physical features of pull requests influence on their lifetime. For instance, when pull requests have a large number of commits, the chances for lengthy lifetimes increase in 92%. The chances for lengthy lifetimes also increase in the order of 54% when multiple files are modified by a sole pull request;

c) The files modified by a pull request and their locations influence on its lifetime. This finding discloses software components that are sensitive and tend to lead to pull requests with lengthy lifetimes and reduced chances of acceptance;

d) Features associated with the contributor impact the pull request lifetime. The lack of experience with the pull-based paradigm increases the chances of lengthy lifetime in up to 85%. When the requester is a core team member, his or her pull requests have 118% more chance to delay. The social relationships also have impact on lifetime. For instance, pull requests from contributors who have no followers on GitHub have 25% more chances of delay;

e) Social features relative to pull request review also exert influence on lifetime. A large number of comments during pull request evaluation may increase the chances of delay in up to 162%.

4) **The identification of factors that influence on the assignment of reviewers for pull requests in open-source projects** – We extracted 22,523 pull requests and identified some patterns that reveal characteristics that influence the reviewer assignment. We discovered that the size of the contribution, requester's lack of experience, social relationship, and recent activity on an artifact influence the reviewer assignment. Understanding these factors is crucial to better devise recommendation tools and provide adequate support to project managers. Such factors may reveal possible conflicts of interest between requesters and reviewers, imbalanced workloads among reviewers, and code sections having only few specialists working on them, which represents a risk considering the workload. The replication of such problems in future recommendations is clearly undesired. Comprehending the nature of reviewer assignment to pull requests is the first step to identify problems such as those and to determine how often they occur. Our main findings are:

a) Some reviewers have preferences for pull request with less commits (with up to 22% more chance of acceptance) and few files (with up to 88% more chances), whereas others tend to analyze more complex contributions;

b) Some reviewers, in up to 58% of times, analyze pull requests submitted by inexperienced requesters. Others, analyze pull requests sent by core team members in up to 71% of times;

c) Some reviewers have more chances (up to 26 times more) of analyzing pull requests submitted by contributors with similar or the same expertise. Others have more chances (up to 59 times more) of evaluating pull requests sent by specific requesters;

d) Some reviewers have more chances (up to 21 times more) of analyzing pull requests that contain files that have been recently modified. Others have more chances (almost 4 times more) of analyzing pull requests that changed specific files and directories.

5) **The exploration of association rules extraction as a technique capable of revealing knowledge on pull requests, offering measures to evaluate the validity of the discovered patterns** - The use of this technique as the core of this work's methodology brought forth the ability to discover implicit information in the data being analyzed, which could have been ignored should a manual exploratory analysis were conducted. This way, association rules represented an effective technique to explore pull request data. Besides, the *Support*, *Confidence*, and *Lift* measures evidence that the patterns found are trustworthy, allowing for an evaluation on the extent of the strength of the uncovered patterns.

Although we have highlighted the implications of our results on the chapters, we summarize here some guidelines for core teams (See Table 7.1) and requesters (See Table 7.2).

Table 7.1: Guidelines for core teams.

Scenarios		
Acceptance	**Lifetime**	**Assigning Reviewers**
Create best practices of collaboration, considering that newcomers sometimes have difficulty understanding how to file pull requests.	Prioritize review of pull requeststhat touch critical areas.	Avoid load unbalance by distributing pull request of different sizes to different reviewers.
Critical regions (locality) of the project should be identified. It is important to create recommendations for contributions filed in these areas.	Define policies for analysis of pull requests filed by newcomers, aiming at improving their involvement with the project.	Create reviewer recommendation tools.
	Define criteria for reviewing pull requests and avoiding delays in the process.	Avoid code owners, thereby increasing the overall understanding of the core team.

Table 7.2: Guidelines for requesters.

Scenarios	
Acceptance	**Lifetime**
Submit simple pull requests, i.e., with only 1 commit.	Submit pull requests with few commits and few files in order to be analyzed in less time and have more the chances of acceptance.
Submit pull requests clear and precise, avoiding much discussion during the evaluation.	Pay attention to pull requests filed in critical locality once can also influence delays.
Pay attention to pull requests in critical regions of the project. They should receive special attention to increase the chances of acceptance.	Do not give up if your first pull request is delayed. This lifetime tends to decrease as the number of contributions increases.

7.2 Future Work

Considering the findings of this thesis, in this Section we suggest the development of some future work.

We propose the fulfillment of studies that consider data derived from source code, i.e., the content of modified files. In this perspective, it would be possible to identify how the modified classes, code refactorings, and/or insertion/removal of complex functions affect the acceptance, lifetime, and assignment of developers. This research has the potential to increase the understanding of the nature of pull requests. In addition, one could observe files that are frequently modified together and to verify the influence of this phenomenon on scenarios investigated in this thesis.

Another proposal is the replication of the experiments using software metrics. In this scenario, one could extract software metrics from pull requests, which quantify some aspects of the software product, for example, cohesion, coupling, and complexity. Then, one could verify the influence of such metrics in the acceptance/rejection, lifetime, and reviewer assignment scenarios. Additionally, one could relate these patterns to those already identified by our studies.

Other aspects to be considered as future work are the pull requests submission chronology and the requesters' geographic data. GitHub is able to provide data that indicates the day and time a pull request was submitted and where the requester is located. In this context, a question arises: is there an effective correlation between time of submission, requester location, and acceptance, lifetime, and reviewer assignment? This could reveal rules that indicate, for example, the propensity to accept/reject pull requests submitted on the weekends, just before holydays, or from specific cities during summer.

We also propose for future work the evaluation of the variation intensity of the discovered patterns. Through temporal analysis of the measures of interest *Support*, *Confidence*, and *Lift*, one could verify trends in these patterns and classify association rules according their variation intensity. For example, association rules could be classified as stable or unstable. Moreover, some tendency to increase/decrease intensity could provide insights for software teams to adopt practices that strengthen positive patterns and weaken negative ones. This would also allow the evaluation of cause and effect of the actions taken by the core team.

As seen in Chapters 3, 4, 5, and 6, the first pull request of a contributor influences on its acceptance/rejection, lifetime, and reviewer assignment. The *"first_pull"* attribute indicates that a contributor is a newcomer. In this way, a suggestion for future work is to investigate more deeply what are the characteristics present in the pull requests filed by newcomers. One could extract rules with *first_pull* = *"true"* in the antecedent, thus being possible to identify physical factors, social factors, metadata, and the content of these pull requests, increasing understanding of the contributions of developers with newcomer profile.

Another suggestion is the development of tools that are able to extract patterns in real time as the software project evolves, that is, decision support systems for the pull request scenario. In this sense, project managers would have a real-time insight on what patterns exert influence – and how strong this influence is – on acceptance, lifetime, and reviewer assignment, thus being able to work for the strengthening of positive patterns and avoidance of the negative ones.

We suggest performing analyses using other evaluation methods, such as hypothesis testing, with the objective of seeking additional evidence in the investigated scenarios.

Another suggestion is an additional analysis that identifies the minimum number of attributes for a compound rule. In addition, we recommend an analysis in rules with lifts less than zero to identify undesirable patterns that should be avoided.

Although we have performed some qualitative analyses, we did not replicate this kind of analysis in all scenarios. We recommend for future work the development of studies that allow a deeper understanding of the discovered patterns, especially to justify complex issues revealed by hybrid rules.

Finally, we recommend studies that evaluate the usefulness of the discovered patterns. We suggest the evaluation of projects in which the core team based their actions on the findings discussed in this thesis. In this way, it is possible to verify how replicable is the knowledge extraction about pull requests and the general comprehensiveness of results.

References

[1] CHACON, S.; STRAUB, B. *Pro Git*. 3rd. ed. San Francisco, CA, USA: Apress, 2009. 107–142 p. ISBN 978-1-4302-1833-3, 978-1-4302-1834-0. Disponível em: <http://dx.doi.org/10.1007/978-1-4302-1834-0_5>.

[2] GUTWIN, C.; PENNER, R.; SCHNEIDER, K. Group awareness in distributed software development. In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. New York, USA: ACM, 2004. (CSCW'04), p. 72–81. ISBN 1-58113-810-5. Disponível em: <http://doi.acm.org/10.1145/1031607.1031621>.

[3] JIMÉNEZ, M.; PIATTINI, M.; VIZCAÍNO, A. Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering*, Hindawi Publishing Corp., New York, United States, v. 2009, p. 3:1–3:16, jan. 2009. ISSN 1687-8655. Disponível em: <http://dx.doi.org/10.1155/2009/710971>.

[4] GOUSIOS, G.; PINZGER, M.; DEURSEN, A. An exploratory study of the pull-based software development model. In: *Proceedings of the 36th International Conference on Software Engineering*. New York, USA: ACM, 2014. (ICSE'14), p. 345–355. ISBN 978-1-4503-2756-5. Disponível em: <http://doi.acm.org/10.1145/2568225.2568260>.

[5] SOARES, D. M.; LIMA JUNIOR, M.L.; MURTA, L.; PLASTINO, A. Rejection factors of pull requests filed by core team developers in software projects with high acceptance rates. In: *Proceedings of the IEEE 14th International Conference on Machine Learning and Applications (ICMLA'15)*. Miami, USA: IEEE, 2015. p. 960–965.

[6] TSAY, J.; DABBISH, L.; HERBSLEB, J. Influence of social and technical factors for evaluating contribution in GitHub. In: *Proceedings of the 36th International Conference on Software Engineering*. New York, USA: ACM, 2014. (ICSE'14), p. 356–366. ISBN 978-1-4503-2756-5. Disponível em: <http://doi.acm.org/10.1145/2568225.2568315>.

[7] RAHMAN, M. M.; ROY, C. K. An insight into the pull requests of GitHub. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, USA: ACM, 2014. (MSR'14), p. 364–367. ISBN 978-1-4503-2863-0. Disponível em: <http://doi.acm.org/10.1145/2597073.2597121>.

[8] PADHYE, R.; MANI, S.; SINHA, V. S. A study of external community contribution to open-source projects on GitHub. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, USA: ACM, 2014. (MSR'14), p. 332–335. ISBN 978-1-4503-2863-0. Disponível em: <http://doi.acm.org/10.1145/2597073.2597113>.

[9] YU, Y.; WANG, H.; FILKOV, V.; DEVANBU, P.; VASILESCU, B. Wait for it: Determinants of pull request evaluation latency on github. In: *Proceedings of*

the IEEE/ACM 12th Working Conference on Mining Software Repositories. Florence, Italy: ACM, 2015. (MSR'15), p. 367–371. ISSN 2160-1852. Disponível em: <http://ieeexplore.ieee.org/document/7180096/>.

[10] YU, Y.; WANG, H.; YIN, G.; LING, C. X. Reviewer recommender of pull-requests in github. In: *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*. Victoria, Canada: IEEE, 2014. (ICSME'14), p. 609–612. ISSN 1063-6773. Disponível em: <http://ieeexplore.ieee.org/abstract/document/6976151/>.

[11] JúNIOR, M. L. de L.; SOARES, D. M.; PLASTINO, A.; MURTA, L. Developers assignment for analyzing pull requests. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2015. (SAC '15), p. 1567–1572. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695884>.

[12] JIANG, J.; HE, J.-H.; CHEN, X.-Y. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology*, Springer, v. 30, n. 5, p. 998–1016, 2015. Disponível em: <https://link.springer.com/article/10.1007/s11390-015-1577-3>.

[13] YING, H.; CHEN, L.; LIANG, T.; WU, J. Earec: leveraging expertise and authority for pull-request reviewer recommendation in github. In: ACM. *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*. Austin, USA, 2016. (ACM'16), p. 29–35. Disponível em: <http://dl.acm.org/citation.cfm?id=2897660>.

[14] RAHMAN, M. M.; ROY, C. K.; COLLINS, J. A. Correct: Code reviewer recommendation in github based on cross-project and technology experience. In: *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA: ACM, 2016. (ICSE '16), p. 222–231. ISBN 978-1-4503-4205-6. Disponível em: <http://doi.acm.org/10.1145/2889160.2889244>.

[15] KOTSIANTIS, S.; KANELLOPOULOS, D. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, v. 32, n. 1, p. 71–82, 2006. Disponível em: <http://www.csis.pace.edu/ ctappert/dps/d861-13/session2-p1.pdf>.

[16] HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. 3rd. ed. Rio de Janeiro, Brasil: Elsevier, 2011.

[17] FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P.; UTHURUSAMY, R. *Advances in Knowledge Discovery and Data Mining*. The MIT Press, 1996. Disponível em: <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0262560976>.

[18] GOUSIOS, G.; SPINELLIS, D. Ghtorrent: Github's data from a firehose. In: *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2012. (MSR '12), p. 12–21. ISBN 978-1-4673-1761-0. Disponível em: <http://dl.acm.org/citation.cfm?id=2664446.2664449>.

[19] AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules in large databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. p. 487–499. ISBN 1-55860-153-8. Disponível em: <http://dl.acm.org/citation.cfm?id=645920.672836>.

[20] NUNES DA SILVA, DANIEL A. *WekaPAR: uma extensão da ferramenta WEKA para auxiliar o pós-processamento de regras de associação*. Rio Branco, Brazil: UFAC.

[21] SOARES, D. M.; JúNIOR, M. L. de L.; MURTA, L.; PLASTINO, A. Acceptance factors of pull requests in open-source projects. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2015. (SAC '15), p. 1541–1546. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695856>.

[22] ESTUBLIER, J. Software configuration management: A roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. New York, USA: ACM, 2000. p. 279–289. ISBN 1-58113-253-0. Disponível em: <http://doi.acm.org/10.1145/336512.336576>.

[23] CONRADI, R.; WESTFECHTEL, B. Version models for software configuration management. *ACM Computing Surveys*, v. 30, n. 2, p. 232–282, jun. 1998. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/280277.280280>.

[24] PRESSMAN, R.; MAXIM, B. *Engenharia de Software*. 8rd. ed. Porto Alegre, Brazil: McGraw Hill Brasil, 2016.

[25] BROSCH, P.; KAPPEL, G.; LANGER, P.; SEIDL, M.; WIELAND, K.; WIMMER, M. An introduction to model versioning. In: BERNARDO, M.; CORTELLESSA, V.; PIERANTONIO, A. (Ed.). *Formal Methods for Model-Driven Engineering*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, 7320). p. 336–398. ISBN 978-3-642-30981-6, 978-3-642-30982-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-30982-3_10>.

[26] HASSAN, A. E.; XIE, T. Mining software engineering data. In: *Proceedings of the 32th ACM/IEEE International Conference on Software Engineering - Volume 2*. New York, USA: ACM, 2010. p. 503–504. ISBN 978-1-60558-719-6. Disponível em: <http://doi.acm.org/10.1145/1810295.1810451>.

[27] CAVALCANTI, Y. C.; NETO, P. A. da M. S.; MACHADO, I. d. C.; VALE, T. F.; ALMEIDA, E. S. de; MEIRA, S. R. d. L. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process*, v. 26, n. 7, p. 620–653, jul. 2014. ISSN 2047-7481. Disponível em: <http://onlinelibrary.wiley.com/doi/10.1002/smr.1639/abstract>.

[28] ZIMMERMANN, T.; WEISGERBER, P.; DIEHL, S.; ZELLER, A. Mining version histories to guide software changes. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE'04), p. 563–572. ISBN 0-7695-2163-0. Disponível em: <http://dl.acm.org/citation.cfm?id=998675.999460>.

[29] ZANDSTRA, M. Version control with git. In: *PHP Objects, Patterns, and Practice*. Apress, 2013. p. 365–382. ISBN 978-1-4302-6031-8, 978-1-4302-6032-5. Disponível em: <http://dx.doi.org/10.1007/978-1-4302-6032-5_17>.

[30] OREG, S.; NOV, O. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, v. 24, n. 5, p. 2055–2073, 2008. ISSN 0747-5632. Including the Special Issue: Internet Empowerment. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0747563207001537>.

[31] WITTEN, I. H.; FRANK, E.; HALL, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*. 4rd. ed. Rio de Janeiro, Brazil: Elsevier, 2016. ISBN 9780128043578.

[32] HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter*, v. 11, n. 1, p. 10–18, nov. 2009. ISSN 1931-0145. Disponível em: <http://doi.acm.org/10.1145/1656274.1656278>.

[33] KROGH, G. V.; HAEFLIGER, S.; SPAETH, S.; WALLIN, M. W. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, v. 36, n. 2, p. 649–676, 2012.

[34] SILVA, M. C. O.; VALENTE, M. T.; TERRA, R. Does technical debt lead to the rejection of pull requests? *CoRR*, abs/1604.01450, 2016. Disponível em: <http://arxiv.org/abs/1604.01450>.

[35] KIKAS, R.; DUMAS, M.; PFAHL, D. Using dynamic and contextual features to predict issue lifetime in github projects. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. New York, USA: ACM, 2016. (MSR '16), p. 291–302. ISBN 978-1-4503-4186-8. Disponível em: <http://doi.acm.org/10.1145/2901739.2901751>.

[36] YU, Y.; WANG, H.; YIN, G.; WANG, T. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, v. 74, p. 204 – 218, 2016. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584916000069>.

[37] TONG, H.; FALOUTSOS, C. Center-piece subgraphs: Problem definition and fast solutions. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2006. (KDD '06), p. 404–413. ISBN 1-59593-339-5. Disponível em: <http://doi.acm.org/10.1145/1150402.1150448>.

[38] STEINMACHER, I.; CONTE, T.; GEROSA, M. A.; REDMILES, D. Social barriers faced by newcomers placing their first contribution in open source software projects. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing*. New York, USA: ACM, 2015. (CSCW'15), p. 1379–1392. ISBN 978-1-4503-2922-4. Disponível em: <http://doi.acm.org/10.1145/2675133.2675215>.