What Are They Talking About? Analyzing Code Reviews in Pull-Based Development Model

Article *in* Journal of Computer Science and Technology · November 2017

DOI: 10.1007/s11390-017-1783-2

5 authors, including:

Zhixing Li
National University of Defense Technology
16 PUBLICATIONS **86** CITATIONS
SEE PROFILE

Yue Yu
National University of Defense Technology
77 PUBLICATIONS **1,102** CITATIONS
SEE PROFILE

Tao Wang
National University of Defense Technology
96 PUBLICATIONS **718** CITATIONS
SEE PROFILE

Li ZX, Yu Y, Yin G *et al.* What are they talking about? Analyzing code reviews in pull-based development model. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 32(6): 1060–1075 Nov. 2017. DOI 10.1007/s11390-017-1783-2

What Are They Talking About? Analyzing Code Reviews in Pull-Based Development Model

Zhi-Xing Li, Yue Yu*, *Member*, *CCF*, *ACM*, Gang Yin, *Member*, *CCF*, *ACM*, Tao Wang, *Member*, *CCF*, *ACM* and Huai-Min Wang, *Fellow*, *CCF*, *Member*, *ACM*

College of Computer, National University of Defense Technology, Changsha 410073, China

E-mail: {lizhixing15, yuyue, yingang, taowang2005, hmwang}@nudt.edu.cn

Received April 21, 2017; revised October 12, 2017.

Abstract Code reviews in pull-based model are open to community users on GitHub. Various participants are taking part in the review discussions and the review topics are not only about the improvement of code contributions but also about project evolution and social interaction. A comprehensive understanding of the review topics in pull-based model would be useful to better organize the code review process and optimize review tasks such as reviewer recommendation and pull-request prioritization. In this paper, we first conduct a qualitative study on three popular open-source software projects hosted on GitHub and construct a fine-grained two-level taxonomy covering four level-1 categories (code correctness, pull-request decision-making, project management, and social interaction) and 11 level-2 subcategories (e.g., defect detecting, reviewer assigning, contribution encouraging). Second, we conduct preliminary quantitative analysis on a large set of review comments that were labeled by TSHC (a two-stage hybrid classification algorithm), which is able to automatically classify review comments by combining rule-based and machine-learning techniques. Through the quantitative study, we explore the typical review patterns. We find that the three projects present similar comments distribution on each subcategory. Pull-requests submitted by inexperienced contributors tend to contain potential issues even though they have passed the tests. Furthermore, external contributors are more likely to break project conventions in their early contributions.

Keywords pull-request, code review, review comment

1 Introduction

The pull-based development model is becoming increasingly popular in distributed collaboration for open-source software (OSS) development[1-4]. On GitHub① alone, the largest social-coding community, nearly half of collaborative projects (already over 1 million[4] in January 2016) adopted this model. In pull-based development, any contributor can freely fork (i.e., clone) an interesting public project and modify the forked repository locally (e.g., fixing bugs and adding new features) without asking for the access to the central repository. When the changes are ready to merge back to the master branch, the contributors submit a pull-request and then a rigorous code review process is performed before the pull-request gets accepted.

Code review is a communication channel where integrators, who are core members of a project, can express their concern for the contribution[3,5-6]. If they doubt the quality of a submitted pull-request, integrators make comments that ask the contributors to improve the implementation. With pull-requests becoming increasingly popular, most large OSS projects allow for crowd sourcing of pull-request reviews to a large number of external developers[7] to reduce the workload of integrators. These external developers are interested in these projects and are concerned about their development. After receiving the review comments, the

Regular Paper

Special Section on Software Systems 2017

This work was supported by the National Key Research and Development Program of China under Grant No. 2016YFB1000805 and the National Natural Science Foundation of China under Grant Nos. 61432020, 61303064, 61472430 and 61502512.

*Corresponding Author

①https://github.com/, Oct. 2017.

contributor usually responds positively and updates the pull-request for another round of review. Thereafter, the responsible integrator makes a decision to accept or reject the pull-request by taking all judgments and changes into consideration.

Previous studies[8-10] have shown that code review, as a well-established software quality practice, is one of the most significant stages in software development. It ensures that only high-quality code changes are accepted, based on the in-depth discussions among reviewers. With the evolution of collaboration tools and environment[11-12], development activities in the communities like GitHub are more transparent and social. Contribution evaluation in the social coding communities is more complex than simple acceptance or rejection[5]. Various participants are taking part in the discussions and the discussion topics are not only about the improvement of code contributions but also about project evolution and social interaction. A comprehensive understanding of the review topics in pull-based model would be useful to better organize the social code review process and optimize review tasks such as reviewer recommendation[7,13] and pull-request prioritization[14].

Several similar studies have been conducted on analyzing reviewers' motivation and issues raised in code reviews. Bacchelli and Bird[15] explored the motivation, outcome and challenge in code reviews based on an investigation of developers and projects at Microsoft. They found that although defect finding is still the major motivation in code reviews, defect related comments comprise a small proportion. Their study is focused on commercial projects and the reviewers of these projects are mainly the employees of Microsoft. However, projects using the pull-based model in GitHub are developed in a more transparent and open environment and code reviews are executed by the community users. Community reviewers may serve for different organizations, use the projects for various purposes, and hold specific consideration in code review processes. Therefore different review topics may exist in the pull-based development model compared with those in the commercial development model. Tsay *et al.*[5] explored issues raised around code contributions in GitHub. They reported that reviewers discussed on both the appropriateness of the contribution proposal and the correctness of the implemented solution. They only analyzed comments of highly discussed pull-requests which have extended discussions. According to the statistics in their dataset, however, each pull-request gets 2.6 comments in average and only about 16% of the pull-requests have extended discussions. The small proportion of explored pull-requests may result in bias in the experimental results and constrain the generalization of their findings. Therefore, we tried to revisit the review topics in the pull-based development model and conduct qualitative and quantitative analysis based on a large-scale dataset of review comments.

In this paper, we first conduct a qualitative study on three popular open-source software projects hosted on GitHub and construct a fine-grained taxonomy covering 11 categories (e.g., defect detecting, reviewer assigning, contribution encouraging) for the review comments generated in the discussions. Second, we perform preliminary quantitative analysis on a large set of review comments that are labeled by TSHC, a two-stage hybrid classification algorithm, which is able to automatically classify review comments by combining rule-based and machine-learning (ML) techniques.

The main contributions of this study include the following.

• A fine-grained and multilevel taxonomy for review comments in the pull-based development model is provided in relation to technical, management, and social aspects.

• A high-quality manually labeled dataset of review comments, which contains more than 5 600 items, can be accessed via a web page② and used in further studies.

• A high-performance automatic classification model for review comments of pull-requests is proposed. The model leads to a significant improvement in terms of the weighted average F-measure, i.e., 9.2% in Rails, 5.3% in Elasticsearch, and 7.2% in Angular.js, compared with the text-based method.

• Typical review patterns are explored, which may provide implications for reviewers and collaborative development platforms to better organize the code review process.

The rest of this paper is organized as follows. Section 2 provides the research background and introduces the research questions. In Section 3, we elaborate the approach of our study. Section 4 lists the research results, while Section 5 discusses several threats to the validity of our study. Section 6 concludes this paper and gives an outlook to the future work.

②https://www.trustie.net/projects/2455, Oct. 2017.

2 Background and Related Work

2.1 Pull-Based Development Model

In GitHub, a growing number of developers contribute to the open source projects by using the pull-request mechanism[2]. As illustrated in Fig.1, a typical contribution process based on the pull-based development model in GitHub involves the following steps.

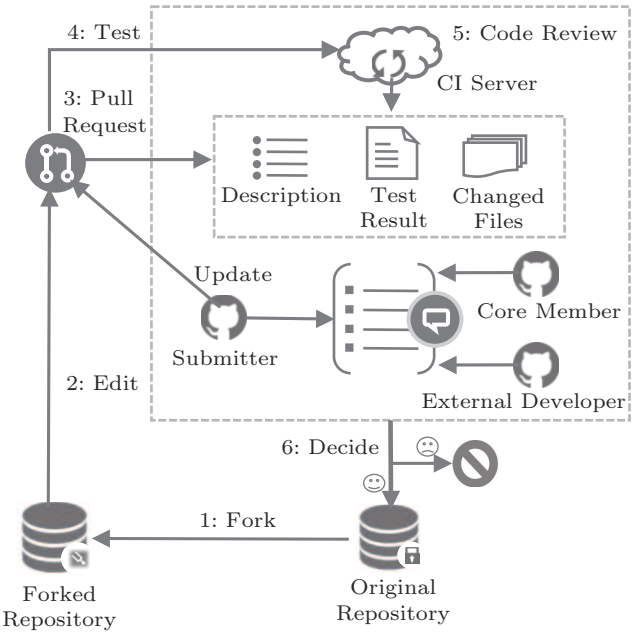


Fig.1. Pull-based work-flow on GitHub.

Fork. A contributor can find an interesting project by following several well-known developers and watching their projects. Before contributing, the contributor has to fork the original project.

Edit. After forking, the contributor can edit locally without disturbing the main branch in the original repository. He/she is free to do whatever he/she wants, such as implementing a new feature or fixing bugs according to the cloned repository.

Pull Request. When the work is finished, the contributor submits the changed codes from the forked repository to its source by a pull-request. Except for commits, the submitter needs to provide a title and description to elaborate on the objective of the pull-request.

Test. Several reviewers play the role of testers to ensure that the pull-request does not break the current runnable state. They check the submitted changes by manually running the patches locally or through an automatic manner with the help of continuous integration (CI) services.

Review. All developers in the community have the chance to discuss the pull-request in the issue tracker, with the existence of pull-request description, changed files, and test result. After receiving the feedback from reviewers, the contributor updates his/her pull-request by attaching new commits for another round of code review.

Decide. A responsible manager of the core team considers all the opinions of reviewers and merges or rejects the pull-request.

Although research on pull-requests is in its early stage, several relevant studies have been conducted in terms of exploratory analysis, priority determination, reviewer recommendation, etc. Gousios *et al.*[2,16] conducted a statistical analysis of millions of pull-requests from GitHub and analyzed the popularity of pull-requests, the factors affecting the decision on a pull-request, and the time to merge a pull-request. Their research result implies that there is no difference between core members and outside contributors in getting their pull-requests accepted and only 13% of the pull-requests are rejected for technical reasons. Tsay *et al.*[8] examined how social and technical information is used to evaluate pull-requests. They found that reviewers take into account of both the contributors' technical practices and the social connection between contributors and reviewers. Yu *et al.*[9,17] conducted a quantitative study, and discovered that latency is a complex issue to explain adequately, and CI is a dominant factor for the process, which even changes the effects of some traditional predictors. Yu *et al.*[7,13] proposed approaches to automatically recommend potential pull-request reviewers, which apply Random Forest algorithm and Vector Space Model respectively. van der Veen *et al.*[14] presented PRioritizer, a prototype pull-request prioritization tool, which works to recognize the top pull-requests the core members should focus on.

2.2 Code Review

Code review is employed by many software projects to examine the changes made by others in source codes, find potential defects, and ensure software quality before they merge back[18]. Traditional code review, which is also well known as code inspection proposed by Fagan[19], has been performed since the 1970s[15]. The inspection process consists of well-defined steps which are executed one by one in group meetings[19]. Many studies[20] have proven the value of code inspection in software development. However, its cumbersome and

synchronous characteristics have hampered its universal application in practice[15]. On the other hand, with the evolution of software development model, this approach is also usually misapplied according to Rigby *et al.*'s exploration[21].

With the occurrence and development of version control systems and collaboration tools, modern code review (MCR)[22] is adopted by many software companies and teams. Different from the formal code inspection, MCR is a lightweight mechanism that is less time-consuming and supported by various tools. Rigby and Storey[22] explored the MCR process in open source communities. They performed case studies on GCC (GNU Compiler Collection), Linux, and other projects and found several code review patterns and a broadcast-based code review style used by Apache. Basum *et al.*[23] analyzed MCR practice in commercial software development teams in order to improve the use of code reviews in industry. While the main motivation for code review was believed to be finding defects to control software quality, recent research[15] has revealed additional expectations, including knowledge transfer, increased team awareness, and creation of alternative solutions to problems. Moreover, other studies[22-24] also investigated the factors that influence the outcomes of code review process.

In recent years, collaboration tools have evolved with social media[11-12]. Especially, GitHub integrates the function of code review in the pull-based model and makes it more transparent and social[11]. The evaluation of pull-requests in GitHub is a form of MCR[15]. The way to give voice on pull-request evaluation is to participate in the discussion and leave comments. Tsay *et al.*[5] analyzed highly discussed pull-requests which have extended discussions and explored evaluating code contributions through discussion in GitHub. Georgios *et al.*[3] investigated the challenges faced by the integrators in GitHub by conducting a survey which involves 749 integrators.

Two kinds of comments can be identified by their positions in the discussions: issue comments (i.e., general comments) as a whole about the overall contribution and inline comments for specific lines of code in the changes[7,25]. Actually, there is another kind of review comments in GitHub: committing comments which are commented on the commitments of pull-requests[26]. Among these three kinds of comments, general comments and inline comments account for the vast majority of all the comments, and the number of committing comments is extremely small. Consequently, we only focus on general comments and inline comments and do not take the committing comments into consideration.

As can be seen from Fig.2, all the review comments corresponding to a pull-request are displayed and ordered primarily by creation time. Issue comments are directly visible, while inline comments is folded by default and will be shown when the toggle button is clicked.

Fig.2. Example comments on GitHub.

Due to the transparent environment, a large number of external developers, who are concerned about the development of an open-source project, are allowed to participate in the evaluation of any pull-request of the public repository. All the participants can ask the submitter to bring out the pull-request more clearly or improve the solution. The prior study[2] also reveals that in most projects, more than half of the participants are external developers, while the majority of the comments come from core team members. The diversity of participants and their different concerns in pull-request evaluation produces various types of review comments which cover a wide range of topics from solution detail to contribution appropriateness.

2.3 Research Questions

In this paper, we focus on analyzing the review comments in GitHub. Although prior work[3,5,15] has identified several motivations and issues raised in code reviews, we believe that the review topics involved in the pull-based development model are not yet well understood, especially that the underlying taxonomy of review topics has not been identified. Consequently, our first question is the following.

*RQ*1. What is the taxonomy for review comments on pull-requests?

Furthermore, we would like to quantitatively analyze review topics and obtain more solid knowledge.

Therefore, we need a large scale of labeled review comments to support the quantitative analysis which leads to the second question.

*RQ*2. Is it possible to automatically classify review comments according to the defined taxonomy?

With a large set of labeled comments being available, we try to conduct quantitative analysis and explore the typical review patterns such as what the most comments are talking about and how contributors raise various kinds of issues in pull-requests. Therefore, our last research question is as follows.

*RQ*3. What are the typical review patterns in the pull-based model?

3 Approach

3.1 Approach Overview

The goal of our work is to investigate the code review practice in GitHub. we aim to qualitatively and quantitatively analyze review topics in the pull-based development model. As illustrated in Fig.3, our research approach consists of the following steps.

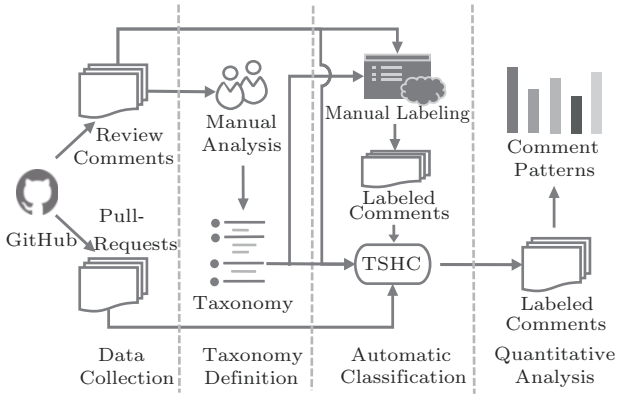


Fig.3. Overview of our approach.

1) *Data Collection*. In our prior work, we collected 4 896 projects that have most number of pull-requests on GitHub. This dataset was crawled through the official API offered by GitHub and updated in the current study. In addition, we also used the data released by GHTorrent③.

2) *Taxonomy Definition*. We constructed a fine-grained multilevel taxonomy for the review comments in the pull-based development model.

3) *Automatic Classification*. We proposed the TSHC algorithm, which automatically classifies review comments using rule-based and ML techniques.

4) *Quantitative Analysis*. We did a preliminary quantitative analysis on a large set of review comments which were labeled by the TSHC algorithm.

3.2 Dataset

The experiments in the paper are conducted on three representative open source projects, namely Rails, Elasticsearch and Angular.js, which make heavy use of pull-requests in GitHub. The dataset is composed of two sources: GHTorrent MySQL dump released in June 2016④ and our own crawled data⑤ from GitHub. From GHTorrent, we can get a list of projects together with their basic information such as programming language, hosting time, the number of forks, and the list of pull-requests. For our own dataset, we have crawled the text content of pull-requests (i.e., title and description) and review comments according to the URLs provided by GHTorrent. Finally, the two sources are linked by the ID of projects and pull-request number. Table 1 lists the key statistics of the dataset.

We studied on original projects (i.e., not forked from others) written in Ruby, Java and JavaScript, which are three of the most popular languages⑥ on GitHub. The three selected projects are hosted on GitHub at an early time and applied in different areas: Rails is used to build websites, Elasticsearch acts as a search server, and Angular.js is an outstanding front-end development framework. This ensures the diversity of experimental projects, which is needed to increase the generalizability of our work. The number of stars, the number of forks, and the number of contributors indicate the hotness of a project. Starring is similar to the function of a bookmarking system which informs users of the latest activities of projects that they have starred.

In total, our dataset contains 27 339 pull-requests and 147 367 review comments.

3.3 Taxonomy Definition

Previous work has studied the challenges faced by pull-request reviewers and the issues introduced by pull-request submitters[3,5]. Inspired by their work, we decide to comprehensively observe the topics of code re-

③http://ghtorrent.org, Oct. 2017.

④http://ghtorrent-downloads.ewi.tudelft.nl/mysql/mysql-2017-06-01.tar.gz, Oct. 2017.

⑤https://www.trustie.net/boards/6297/topics/44132, Oct. 2017.

⑥https://github.com/blog/2047-language-trends-on-github, Oct. 2017.

Table 1. Dataset of Our Experiments

Project	Language	Application Area	Hosted_at	#Star	#Fork	#Contr	#PR	#Comnt
Rails	Ruby	Web framework	May 20, 2009	33 906	13 789	3 194	14 648	75 102
Elasticsearch	Java	Search server	Feb. 8, 2010	20 008	6 871	753	6 315	38 930
Angular.js	JavaScript	Front-end framework	Jan. 6, 2010	54 231	26 930	1 557	6 376	33 335

Note: Contr: contributor; PR: pull-request; Comnt: comment; #: number of.

views in depth rather than merely to focus on technical and nontechnical perspectives.

We conducted a card sort[15] to determine the taxonomy schema, which was executed manually through an iterative process of reading and analyzing review comments randomly collected from the three projects. The following steps were executed to define the taxonomy.

1) First, we randomly selected 900 review comments (300 for each project).

2) Two main participants independently conducted card sorts on 70% of the comments. They firstly labeled each selected comment with a descriptive message. The labeled comments were then divided into different groups according to their descriptive messages. Through a rigorous analysis of existing literature and their own experience with working and analyzing the pull-based model in the last two years, they identified their respective draft taxonomies.

3) They then met and discussed their identified draft taxonomies. When they agreed with each other, the initial taxonomy was constructed.

4) Another 10 participants were invited to help varify and improve the taxonomy by examining another 10% of the comments. After some refinement of category definition and adjustment of taxonomy hierarchy was executed, the final taxonomy was determined.

5) Finally, the two main participants independently classified the remaining 20% of the comments into the taxonomy. We used one of the most popular reliability measurements, which is percent agreement, to calculate their reliability. We found they agreed on 97.8% (176/180) of the labeling of comments.

In the above process, the two main participants (i.e., the first two authors of the paper) have five years' and eight years' experience in software development respectively. Both of them are interested in academic research in empirical software engineering and social coding. Especially, the second participant has four-year research experience in the pull-based development model. Another 10 participants also work at our research team including postgraduates, Ph.D. candidates, and project developers.

3.4 Automatic Classification

After the taxonomy of review topics is defined, we would like to conduct quantitative study on a large scale of labeled review comments which are expected to be classified by an automatic approach. First of all, we need to manually label a set of comments to train the automatic classification model. And then, we use the classification model to automatically label a large dataset of comments.

3.4.1 Manual Labeling

For each project, we randomly sampled 200 pull-requests (each of which gets more than 0 comment) per year from 2013 to 2015. Overall, 1 800 distinct pull-requests and 5 645 review comments are sampled. According to the defined taxonomy, we manually classified the sampled review comments. We built an online labeling platform (OLP) which was deployed on the public network and offered a web-based interface. OLP is helpful to reduce the extra burden on the labeling executor and ensure the quality of the manual labeling results.

As shown in Fig.4, the main interface of OLP contains three sections.

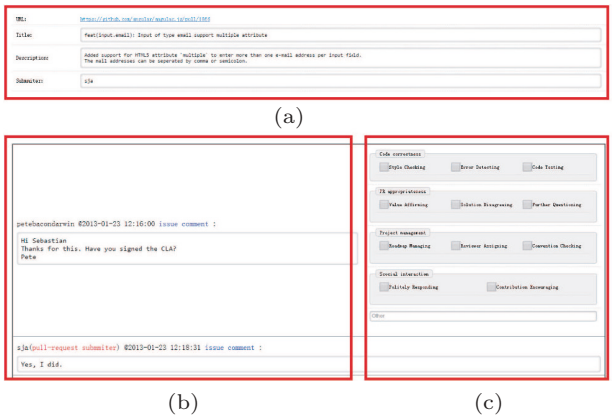


Fig.4. Main interface of OLP. (a) Section 1. (b) Section 2. (c) Section 3.

Section 1. The title, description, and submitter's user name of a pull-request are displayed in this sec-

tion. The hyper-link of the pull-request on GitHub is also shown to make it more convenient to jump to the original web page for a more detailed view and inspection if necessary.

Section 2. All the review comments of a pull-request are listed in this section and ordered by creation time. The user name, comment type (inline comment or issue comment), and creation time of a review comment appear on the top of comment content.

Section 3. Besides a comment, there are candidate labels corresponding to our taxonomy, which can be selected by clicking the check-box next to the label. The use of check-boxes means that multiple labels can be assigned to a comment. Other labels that are not in our list can also be reported by writing free text in the text field named "other".

3.4.2 Two-Stage Hybrid Classification

Fig.5 illustrates the automatic classification model TSHC. TSHC consists of two stages that utilize comments text and other information extracted from comments and pull-requests respectively.

Stage 1. The classification in this stage mainly utilizes the text part of each review comment and produces a vector of possibility (VP), in which each item is the possibility of whether a review comment will be labeled with the corresponding category.

Preprocessing is necessary before formal comment classification. Reviewers tend to reference the source code, hyper-link, or statement of others in a review comment to clearly express their opinions, prove their points, or reply to other people. These behaviors promote the review process but cause a great challenge to comment classification. Words in these reference texts contribute minimally to the classification and even introduce interference. Hence, we transform them into single-word indicators to reduce vocabulary interference and reserve reference information. Specifically, the source code, hyperlink, and statement of others are replaced with "cmmcode", "cmmlink" and "cmmtalk" respectively.

After preprocessing, a review comment is classified by the rule-based technique, which uses inspection rules to match the comment text for each category. Several phrases often appear in the comments of a specific category. For instance, "lgtm" (abbreviation of "looks good to me") is usually used by reviewers to express their satisfaction with a pull-request. This type of phrases is discriminating and helpful in recognizing the category of a review comment. Therefore, we established inspection rules for each category, which are a set of regular expressions abstracted from discriminating phrases. A category label is assigned to a review comment, and the corresponding item in VP is set to 1 if one of its inspection rules matches the comment text. In our method each category gets 7.5 rules in average and only part of the inspection rules and the corresponding matched comments are listed below.

- Example inspection rule 1
  - `(blank|extra) (line|space)s?`
  - "*Please add a new* ***blank line*** *after the include*"
- Example inspection rule 2
  - `(looks|seem)s? (good|great|useful)`
  - "***Looks good*** *to me, the grammar is definitely better.*"
- Example inspection rule 3
  - `(cc:?|wdyt|defer to|\br\?) (@\w+ *)+`

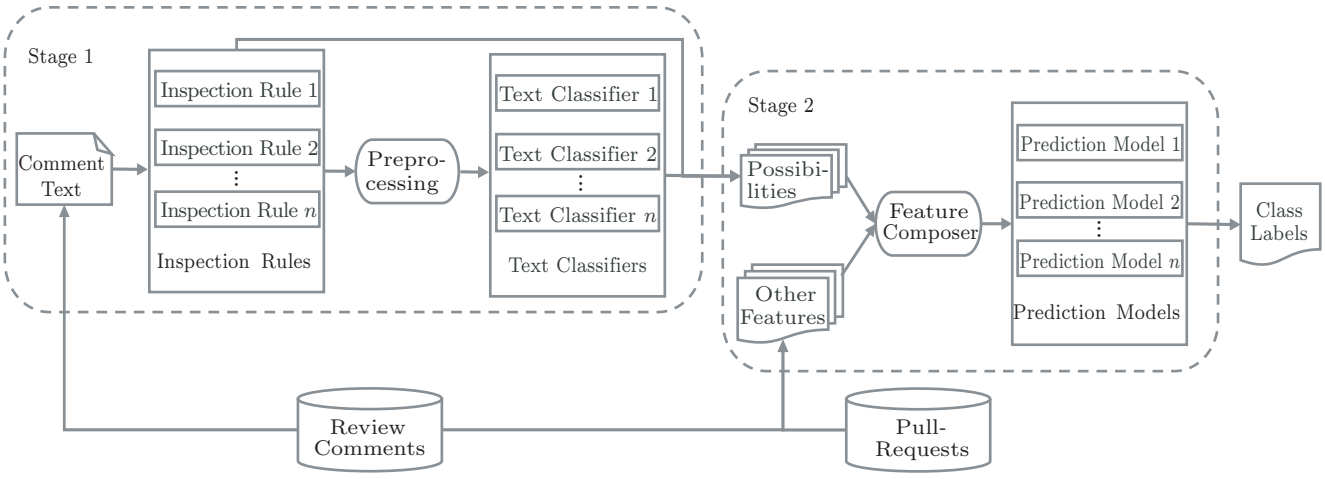


Fig.5. Overview of TSHC.

– "***/cc @fxn*** *can you take a look please?*"

• Example inspection rule 4

– `thanks?|thxs?|:(\w+)?heart:`

– "*@xxx looks good.* ***Thank*** *you for your contribution* ***:yellow_heart:***"

The review comment is then processed by the ML-based technique. ML-based classification is performed with scikit-learn⑦, particularly the support vector machine (SVM) algorithm. The comment text is tokenized and each token is stemmed to the root form. We filtered out punctuations from word tokens and reserved English stop words because we assumed that common words play an important role in short texts, such as review comments. We adopted the TF-IDF (term frequency-inverse document frequency) model[27] to extract a set of features from the comment text and applied the ML algorithm to text classification. A single review comment often addresses multiple topics. Hence, one of the goals of TSHC is to perform multi-label classification. To this end, we construct text classifiers (TCs) for each category with a one-versus-all strategy. For a review comment which has been matched by inspection rule R_i (supposing that n categories of $C_1, C_2, \ldots, C_n$ exist), each TC $(TC_1, TC_2, \ldots, TC_n)$, except for TC_i, is applied and predicts the possibility of this review comment belonging to the corresponding category.

Finally, the VP determined by inspection rules and text classifiers is passed on to the second stage.

Stage 2. The classification in this stage is performed on composed features. Review comments are usually short texts. Our statistics indicates that the minimum, maximum, and average numbers of words contained in a review comment are 1, 1 527, and 32, respectively. The text information contained in a review comment is insufficient to determine its category. Therefore, in addition to the VP generated in stage 1, we also considered the following other features related to review comments.

Comment_length. This feature refers to the total number of characters contained in a comment text after preprocessing. Long comments are likely to argue about pull-request appropriateness and code correctness.

Comment_type. This binary feature indicates whether a review comment is an inline comment or an issue comment. An inline comment tends to talk about the solution detail, whereas an issue comment is likely to talk about other "high-level" issues, such as pull-request decision and project management.

Core_team. This binary feature refers to whether the comment author is a core member of a project or an external contributor. Core members are more likely to pay attention to pull-request appropriateness and project management.

Link_inclusion. This binary feature identifies if a comment includes hyper-links. Hyper-links are usually used to provide evidence when someone insists on a point of view or to offer guidelines when someone wants to help other people.

Ping_inclusion. This binary feature determines if a comment includes ping activity (occurring by the form of "@ user-name").

Code_inclusion. This binary feature denotes if a comment includes the code elements. Comments related to the solution detail tend to contain code elements. However, this assumption does not limit whether the code is a replica of the committed code or a new code snippet written by the reviewer

Ref_inclusion. This binary feature indicates if a comment includes a reference on the statement of others. Such a reference indicates a reply to someone, which probably reflects further suggestion to or disagreement with a person.

Sim_pr_title. This feature refers to the similarity between the text of the comment and the text of the pull-request title (measured by the number of common words divided by the number of union words).

Sim_pr_desc. This binary feature denotes the similarity between the text of the comment and the text of the pull-request description (measured similarly as how Sim_pr_title is computed). Comments with high similarity to the title or description of a pull-request are likely about the solution detail or the value of the pull-request.

Together with the VP passed from stage 1, these features are composed to form a new feature vector to be processed by prediction models. Similar to stage 1, stage 2 provides binary prediction models for each category. In the prediction models, a new VP is generated to represent how likely a comment will fall into a specific category. After iterating the VP, a comment is labeled with class C_i if the i-th vector item is greater than 0.5. If all the items of the VP are less than 0.5, the class label corresponding to the largest possibility will be assigned to the comment. Finally, each comment processed by TSHC is marked with at least one class label.

⑦http://scikit-learn.org/, Oct. 2017.

3.5 Quantitative Analysis

To identify typical review patterns, we used our trained hybrid classification model to classify a large scale of review comments and then examined the review patterns based on this automatically classified dataset. In total, we classified 147 367 comments of 27 339 pull-requests. Based on this dataset, we conducted a preliminary quantitative study. We first explored the distribution of review comments on each category and then we reported some of the findings derived from the distribution.

4 Results

4.1 RQ1: What Is the Taxonomy for Review Comments on Pull-Requests

Table 2 shows the complete taxonomy. We identified four level-1 (L1) categories, namely code correctness (L1-1), PR (pull-request) decision-making (L1-2), project management (L1-3), and social interaction (L1-4), each of which contains more specific level-2 subcategories. For each level-2 subcategory, we present its description together with an example comment. We show short comments or only the key part of long comments to provide a brief display. Compared with previous studies[5,15], our hierarchical taxonomy presents a more systematic and elaborate schema of review topics in the pull-based model. Moreover, we succeeded in identifying a group of review comments related to project management (e.g., road-map managing (L2-7), reviewer assigning (L2-8)) and found them to be proper positions in the final taxonomy.

Also, we would like to point out it is a common phenomenon that a single comment usually covers multiple topics. The following review comments are provided as examples.

Example 1. "Thanks for your contribution! This

Table 2. Complete Taxonomy

Level-1 Category	Level-2 Subcategory	Description & Example
Code correctness (L1-1)	Style checking (L2-1)	Pointing out extra blank lines, improper indention, inconsistent naming convention, etc. e.g., "scissors: this blank line"
	Defect detecting (L2-2)	Figuring out runtime program errors or evolvability defects, etc. e.g., "he default should be 'false'" and "let's extract this into a constant. No need to initialize it on every call."
	Code testing (L2-3)	Demanding submitters to provide test case for changed codes, reporting test results, etc. e.g., "this PR will need a unit test, I'm afraid, before it can be merged."
PR decision-making (L1-2)	Value affirming (L2-4)	Satisfied with the pull-request and agreeing to merge it e.g., "PR looks good to me. Can you ..."
	Solution disagreeing (L2-5)	Rejecting to merge the pull-request for duplicate proposal, undesired features, etc. e.g., "I do not think this is a feature we'd like to accept. ..."
	Further questioning (L2-6)	Confused with the purpose of the pull-request and asking for more details or use cases e.g., "Can you provide a use case for this change?"
Project management (L1-3)	Road-map managing (L2-7)	Stating what type of changes a specific version is expected to merge, etc. e.g., "Closing as 3-2-stable is security fixes only now"
	Reviewer assigning (L2-8)	Pinging other reviewers to review this pull-request e.g., "/cc @fxn can you take a look please?"
	Convention checking (L2-9)	Demanding submitters to squash the commits, formulate the messages, etc. e.g., "...Can you squash the two commits into one? ..."
Social interaction (L1-4)	Politely responding (L2-10)	Expressing thanks for others' contribution, apologizing for mistakes, etc. e.g., "Thank you. This feature was already proposed and it was rejected. See #xxx"
	Contribution encouraging (L2-11)	Agreeing with others' opinions, complimenting others' work, etc. e.g., ":+1: nice one @cristianbica."
Others	Short sentences without clear or exact meaning like "@xxxx will do" and "The same here :)"	

Note: Word beginning and ending with colon like ":scissors:" is a markdown grammar for emoji in GitHub.

looks good to me but maybe you could split your patch in two different commits? One for each issue."

Example 2. "Looks good to me :+1: /cc @steveklabnik @fxn."

In the first comment, the reviewer thanks the contributor (L2-10) and shows his/her satisfaction with this pull-request (L2-4), followed by a request of commit splitting (L2-9). In the second comment, the reviewer expresses that he/she agrees with this change (L2-4), thinks highly of this work (L2-11, :+1: is markdown grammar for emoji "thumb-up"), and assigns other reviewers to ask for more advice (L2-11). With regard to reviewer assignment, reviewers tend to delegate a code review if they are unfamiliar with the change under review.

In addition, we collected 13 exception comments which are labeled with "other" option. These comments mainly fall into the following two groups.

• *Platform-Related.* This kind of comments is related to the platform, i.e., GitHub. Sometimes, developers may be not familiar with the features of GitHub: "I don't understand why GitHub display all my commits."

• *Simple Reply.* Comments of this kind have only a very few words and have not exact significant meaning: "@** no.", "yes" and "done".

On one hand, the number of these comments is relatively small and on the other hand these comments do not provide too much contribution to the code review process. Moreover, the process of taxonomy definition in Subsection 3.3 has made our taxonomy relatively complete and robust for its large-scale sampling, thorough discussion and multiple validation. Therefore, we exclude them from our analysis and do not adjust our taxonomy.

Summary. From the qualitative study, we identified a two-level taxonomy for review comments which consists of four categories in level 1 and 11 subcategories in level 2.

4.2 RQ2: Is It Possible to Automatically Classify Review Comments According to the Defined Taxonomy

In the evaluation, we designed a text-based classifier (TBC) as a comparison baseline which only uses the text content of review comments and does not apply any inspection rule or composed feature. TBC uses the same preprocessing techniques and SVM models as used in TSHC. Classification performance is evaluated through a 10-fold cross-validation, namely splitting review comments into 10 sets, of which nine sets are used to train the classifiers and the remaining set is for the performance test. The process is repeated 10 times. Moreover, the evaluation metrics used in the paper are computed by the following formulas.

$$Prec(\text{L2-}i) = \frac{N_{\text{CC}}(\text{L2-}i)}{N_{\text{TSHC}}(\text{L2-}i)},$$
$$Rec(\text{L2-}i) = \frac{N_{\text{CC}}(\text{L2-}i)}{N_{\text{total}}(\text{L2-}i)},$$
$$F\text{-}M(\text{L2-}i) = \frac{2 \times Prec(\text{L2-}i) \times Rec(\text{L2-}i)}{Prec(\text{L2-}i) + Rec(\text{L2-}i)}.$$

$Prec$(L2-i), Rec(L2-i), and F-M(L2-i) measure the precision, recall and F-measure of each method on category L2-i respectively. N_{total}(L2-i) is the total number of comments of category L2-i in the test dataset, N_{TSHC}(L2-i) is the number of comments that are classfied as L2-i by TSHC, and N_{CC}(L2-i) is the number of comments that have been correctly classified as L2-i.

Table 3 shows the precision, recall, and F-measure provided by different approaches for level-2 subcategories. Our approach achieves the highest precision, recall, and F-measure scores in all categories with only a few exceptions.

To provide an overall performance evaluation, we used the weighted average metric value[28] of all categories according to the proportions of instances in each category. (1) describes the formula to derive the average F-measure. In the equation, the average F-measure is denoted as F_{avg}, the F-measure of category L2-i as f_i, and the number of instances of category L2-i as n_i.

$$F_{\text{avg}} = \frac{\sum_{i=1}^{11} n_i \times f_i}{\sum_{i=1}^{11} n_i}. \quad (1)$$

Table 3 indicates that our approach consistently outperforms the baseline across the three projects. Compared with that in the baseline, the improvement in TSHC running on each project in terms of the weighted average F-measure is 9.2% (from 0.688 to 0.780) in Rails, 5.3% (from 0.767 to 0.820) in Elasticsearch, and 7.2% (from 0.675 to 0.747) in Angular.js. These results indicate that our approach is highly applicable in practice.

Furthermore, we would like to explore whether the rule-based technique or composed features contribute more to the increased performance of TSHC compared with the baseline TBC. Therefore, we designed another two comparative settings.

Table 3. Classification Performance on Level-2 Subcategories

SC	Rails						Elasticsearch						Angular.js					
	TBC			TSHC			TBC			TSHC			TBC			TSHC		
	Prec	*Rec*	*F-M*	*Prec*	*Rec*	*F-M*	*Prec*	*Rec*	*F-M*	*Prec*	*Rec*	*F-M*	*Prec*	*Rec*	*F-M*	*Prec*	*Rec*	*F-M*
L2-1	0.75	0.57	0.66	0.88	0.67	0.78	0.75	0.46	0.61	0.85	0.58	0.72	0.54	0.26	0.40	0.76	0.78	0.77
L2-2	0.63	0.84	0.74	0.71	0.86	0.79	0.67	0.92	0.80	0.77	0.82	0.80	0.65	0.71	0.68	0.69	0.71	0.70
L2-3	0.59	0.46	0.53	0.74	0.78	0.76	0.66	0.55	0.61	0.60	0.52	0.56	0.64	0.63	0.64	0.75	0.77	0.76
L2-4	0.56	0.35	0.46	0.73	0.58	0.66	0.92	0.84	0.88	0.91	0.88	0.90	0.83	0.72	0.78	0.84	0.79	0.82
L2-5	0.50	0.26	0.38	0.63	0.43	0.53	0.65	0.36	0.51	0.80	0.67	0.74	0.63	0.48	0.56	0.61	0.51	0.56
L2-6	0.47	0.21	0.34	0.60	0.36	0.48	0.33	0.11	0.22	0.38	0.26	0.32	0.45	0.51	0.48	0.47	0.49	0.48
L2-7	0.79	0.66	0.73	0.79	0.72	0.76	0.75	0.39	0.57	0.75	0.68	0.72	0.49	0.31	0.40	0.83	0.64	0.74
L2-8	0.83	0.77	0.80	0.98	0.78	0.88	0.57	0.35	0.46	0.88	0.90	0.89	0.35	0.16	0.26	0.96	0.67	0.82
L2-9	0.83	0.76	0.80	0.90	0.81	0.86	0.94	0.57	0.76	0.96	0.76	0.86	0.85	0.76	0.81	0.83	0.82	0.83
L2-10	0.98	0.93	0.96	0.99	0.98	0.99	0.91	0.84	0.88	0.93	0.95	0.94	0.90	0.80	0.85	0.94	0.93	0.94
L2-11	0.80	0.66	0.73	0.89	0.87	0.88	0.87	0.67	0.77	0.92	0.91	0.92	0.93	0.62	0.78	0.98	0.92	0.95
AVG	0.71	0.67	0.69	0.80	0.76	0.78	0.79	0.75	0.77	0.83	0.81	0.82	0.71	0.64	0.67	0.76	0.73	0.75

Note: SC: subcategory; *Prec*: precision; *Rec*: recall; *F-M*: *F*-measure; AVG: average.

• TBC_R ($TBC + Rule$). In this setting, we combined TBC with rule-based technique.

• TBC_CF ($TBC + Composed\ Features$). In this setting, we combined TBC with composed features in stage 2.

Table 4 presents the experimental result. Overall speaking, TBC_R is better than TBC_CF, which means rule-based technique contributes more to the increased performance than composed features in stage 2. Compared with the TBC, TBC_R achieves an improvement of 7.24%, 5.19%, and 8.96% for Rails, Elasticsearch and Angular.JS respectively, while TBC_CF achieves 2.9%, 1.30% and 2.99% respectively.

We further studied the review comments miscategorized by TSHC. And the following is an example of such comments.

Example 3. "While karma does globally install with a bunch of plug-ins, we do need the npm install because without that you don't get the karma-ng-scenario karma plug-in."

TSHC classifies it to error detecting (L2-2), but it is actually a solution disagreeing (L2-5) comment. The reason for this incorrect predication is two-fold, namely the lack of explicit discriminating terms and the specific explanation for rejection. Inspection rule of solution disagreeing (L2-5) is unable to be matched because of the lack of corresponding matching patterns. ML classifiers tend to categorize it into error detecting (L2-2) because the specific explanation of opinion makes it more like a low-level comment about code correctness instead of a high-level one about the pull-request decision.

We attempted to solve this problem by adding factors (e.g., comment_type and code_inclusion) in stage 2 of TSHC, which can help reveal whether a review comment is talking about the pull-request as a whole or the solution detail. Although the additional information improves the classification performance to some extent, it is not sufficient to differentiate the two types of comments. We plan to address the issue by extending the manually labeled dataset and introducing a sentiment analysis.

Summary. Compared with the baseline, TSHC achieves higher performance in terms of the weighted average *F*-measure, namely 0.78 in Rails, 0.82 in Elasticsearch, and 0.75 in Angular.js, which indicates TSHC is applicable in practical automatic classification.

Table 4. Classification Performance of Different Methods

Performance	Rails			Elasticsearch			Angular.js		
	TBC	TBC_R	TBC_CF	TBC	TBC_R	TBC_CF	TBC	TBC_R	TBC_CF
F_{avg}	0.69	0.74	0.71	0.77	0.81	0.78	0.67	0.73	0.69
Improvement (%)	-	7.24	2.90	-	5.19	1.30	-	8.96	3.00

4.3 RQ3: What Are the Typical Review Patterns Among the Reviewers' Discussions

We now discuss some of our preliminary quantitative analysis. We used TSHC to automatically classify all the review comments, based on which we explored the quantitative characteristics of review comments.

4.3.1 Comments Distribution

Fig.6 shows the percentage of the level-2 subcategories. As a general view, the comments distribution over the three projects represent similar patterns, in which most comments are about defect detecting (L2-2), value affirming (L2-4) and politely responding (L2-10). Significantly, defect detecting (L2-2) occupies the first place (Rails: 42.5%, Elasticsearch: 50.6%, Angular.js: 33.1%). This is consistent with previous studies[3,15] which states that the main purpose of code review is to find defects. Moreover, the development of open source projects relies on the voluntary collaborative contribution[29] of thousands of developers. A harmonious and friendly community environment is beneficial to promote contributors' enthusiasm which is critical to the continuous evolution of a project. As a result, a majority of reviewers never hesitate to react positively to others' contribution (L2-10, L2-11) and express their satisfaction with high-quality pull-requests (L2-4).

Actually, we also explored comments distribution based on the manually labeled dataset and found it was similar to that on the automatically labeled dataset. The distribution difference between the two datasets is that the manually labeled dataset has more comments of L2-4 and less comments of L2-2 compared with the automatically labeled dataset. But the difference is slight and does not affect the distribution outline. As a result, we only reported our findings on the automatically labeled dataset which has the larger data size and therefore makes our findings more convincing.

4.3.2 Cost-Free Comments

It is strange that although inspecting code style (L2-1) and testing code (L2-3) cost reviewers very little effort, and do not require too much understanding for code changes, comments of these two categories are relatively less than others. This is somewhat inconsistent with the study conducted by Bacchelli and Bird[15] on Microsoft teams, in which they found code style (called code improvement in their study) is the most frequent outcome of code reviews. The reason for the inconsistence, in our opinion, is the difference of development environment. In GitHub, contributors are in a transparent environment and their activities are visible to everyone[4,8]. Hence, in order to build and maintain good reputations, contributors usually go through their code by themselves before submission and prevent making apparent mistakes[4].

4.3.3 Defects Under Green Tests

We were also curious about why a large percentage of comments focus on detecting defects (L2-2) even though most of the pull-requests did not raise testing issues (L2-3). After examining again those pull-requests which received comments of L2-2 but did not receive comments of L2-3, we found two main factors resulting in this phenomenon.

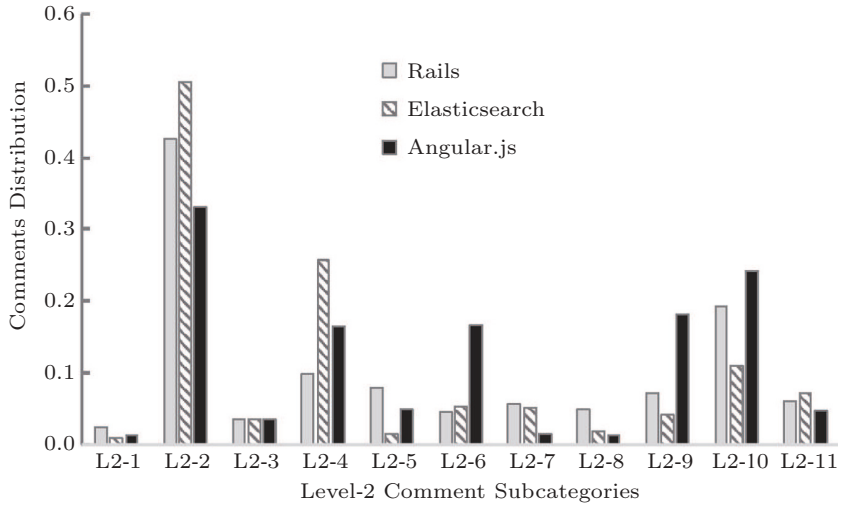


Fig.6. Percentage of the level-2 subcategories.

One factor is contributors' unfamiliarity with the programming language and the APIs, which is shown in the following example comments.

Example 4. "Just this change is not necessary, it'll run validations twice, you can use r.save in the assert call."

Example 5. "This will introduce a memory leakage on system where there are a lot of possible SQLs. Remember symbols are not removed from memory from the garbage collector."

The other one is the lack of development experience of some contributors, which is shown in the following example comments.

Example 6. "This will change the existing behavior and people may be relying on it so I don't think it is safe to change it like this. If we want to change this we need a better upgrade path, like a global option to let people to switch the behavior."

Example 7. "This new class isn't needed — we can reuse an existing one."

These factors tend to produce runnable but "low-quality" codes which are of less elegance or even break the compatibility and introduce potential risks. This reality justifies the necessary of manual code review even although an increasing number of automatic tools are coming into being.

4.3.4 Overlooked Convention

Also most projects set specific development conventions which do not require too much effort to follow, and they are, somehow, overlooked by contributors which can be seen from the following examples.

Example 8. "Thanks! Could you please add '[ci skip]' to your commit message to avoid Travis to trigger a build?"

Example 9. "@xxx PR looks good to me. Can you squash the two commits into one. Fix and test should go there in a commit. Also please add a change log. Thanks."

To better understand this problem, we did a statistical analysis to explore whether this kind of comments (L2-9) are distributed differently on the roles of contributors (core members or external contributors).

We found that pull-requests submitted by external contributors receive more comments of L2-9 than those submitted by core members, which accounts for 83.9%, 60.1%, and 80.7% of the total number in Rails, Elasticsearch, and Angular.js respectively.

Furthermore, we explored how an external developer's contribution experience affects the possibility of his/her pull-requests getting comments of L2-9. Fig.7 shows how many the pull-requests that receive comments of L2-9 are created and accumulated during developers' contribution history. It is clear that a considerable proportion of such pull-requests (68.4% in Rails, 75.1% in Elasticsearch, and 91.9% in Angular.js) have been generated in the first five submissions. Fig.7 illustrates that external contributors are more likely to submit pull-requests that break project conventions in their early contributions. Therefore, it is necessary to make external contributors gain a well understanding of project conventions in the very early stage.

In fact, most project management teams have provided contributors with specific guidelines[8]. It seems, however, that not everyone is willing to go through the tedious specification before contribution. This may inspire GitHub to improve its collaborative mechanism and offer more efficient development tools. For example, it is better to briefly display a clear list of development conventions which are edited by the management team of a project before a developer creates a pull-request to this project. Another alternative solution for GitHub is to provide automatic reviewing tools that can be configured with predefined convention rules and triggered as a new pull-request is submitted.

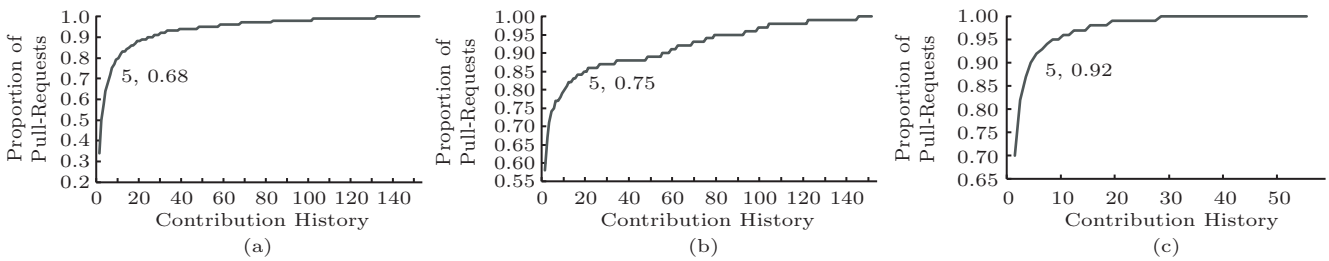


Fig.7. Cumulative distribution of pull-requests. (a) Rails. (b) Elasticsearch. (c) Angular.js.

[8]http://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html, Oct. 2017.

Summary. Most comments are discussing about code correcting and social interactions. External contributors are more likely to break project conventions in their early contributions, and their pull-requests might contain potential issues, even though they have passed the tests.

5 Threats to Validity

In this section, we discuss threats to construct validity, internal validity and external validity, which may affect the results of our study.

Construct Validity. The first threat involves the taxonomy definition since some of the categories could be overlapped or missed. To alleviate this problem, we, together with other experienced developers, randomly selected review comments and classified them according to our taxonomy definition. The verification result in Subsection 3.3 showed that our taxonomy is complete and does not miss any significant categories.

Secondly, manually labeling thousands of review comments is a repetitive, time-consuming and boring task. To get a reliable labeled set, the first author of the paper was assigned to do this job and we tried our best to provide him a pleasant work environment. Moreover, if it has been a long time since the last round of labeling work, the first author would revisit the taxonomy and go through some labeled comments to ensure his accurate and consistent understanding of his job before the next round of labeling work proceeds. It is possible that the incorrect classification of TSHC may affect our findings even when our model achieved high precision of 76%∼83%. However, our preliminary quantitative study is mainly based on the comments distribution on each category, and wrong classification is not likely to alter the distribution significantly; therefore our findings are not easily affected by some extent of incorrect classification.

Internal Validity. There are many machine learning methods that can be used to solve the classification problem and the choice of ML algorithm has a direct impact on the classification performance. We compared several ML algorithms including linear regression classifier, adaBoost classifier, random forest classifier, and SVM, and found that SVM performed better than the others on classifying review comments. Furthermore, we also did some necessary parameter optimization.

External Validity. Our findings are based on the dataset of three open source projects hosted on GitHub. To increase the generalizability of our research, we have selected projects with different programming languages and different application areas. Nevertheless our dataset is a small sample compared with the total number of projects in GitHub. Hence, it is not very sure whether the results can be generalized to all the other projects hosted on GitHub or those which are hosted on other platforms.

6 Conclusions

Code review is one of the most significant stages in pull-based development. It ensures that only high-quality pull-requests are accepted, based on the in-depth discussions among reviewers. To comprehensively understand the review topics in the pull-based model, we first conducted a qualitative study on three popular open-source software projects hosted on GitHub and constructed a fine-grained two-level taxonomy covering four level-1 categories (code correctness, PR decision-making, project management, and social interaction) and 11 level-2 subcategories (e.g., defect detecting, reviewer assigning, contribution encouraging). Second, we did preliminary quantitative analysis on a large set of review comments that are labeled by a two-stage hybrid classification algorithm, TSHC, which is able to automatically classify review comments by combining rule-based and machine-learning techniques. Through the quantitative study, we explored the typical review patterns. We found that the three projects present similar comments distribution on each category. Pull-requests submitted by inexperienced contributors tend to contain potential issues even though they have passed the tests. Furthermore, external contributors are more likely to break project conventions in their early contributions.

Nevertheless, TSHC performs poorly on a few level-2 subcategories. More work could be done in the future to improve it. We plan to address the shortcomings of our approach by extending the manually labeled dataset and introducing sentiment analysis. Moreover, we will try to dig more valuable information (e.g., comment co-occurrence, emotion shift) from the experimental result in the paper and assist core members to better organize the code review process, such as improving reviewer recommendation, contributor assessment, and pull-request prioritization.

References

[1] Barr E T, Bird C, Rigby P C, Hindle A, German D M, Devanbu P. Cohesive and isolated development with branches.

In *Fundamental Approaches to Software Engineering*, De Lara J, Zisman A (eds.), Springer, 2012, pp.316-331.

[2] Gousios G, Pinzger M, van Deursen A. An exploratory study of the pull-based software development model. In *Proc. the 36th Int. Conf. Software Engineering*, May 31-June 7, 2014, pp.345-355.

[3] Gousios G, Zaidman A, Storey M A, van Deursen A. Work practices and challenges in pull-based development: The integrator's perspective. In *Proc. the 37th Int. Conf. Software Engineering*, May 2015, pp.358-368.

[4] Gousios G, Storey M A, Bacchelli A. Work practices and challenges in pull-based development: The contributor's perspective. In *Proc. the 38th Int. Conf. Software Engineering*, May 2016, pp.285-296.

[5] Tsay J, Dabbish L, Herbsleb J. Let's talk about it: Evaluating contributions through discussion in GitHub. In *Proc. the 22nd ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, November 2014, pp.144-154.

[6] Marlow J, Dabbish L, Herbsleb J. Impression formation in online peer production: Activity traces and personal profiles in GitHub. In *Proc. Conf. Computer Supported Cooperative Work*, February 2013, pp.117-128.

[7] Yu Y, Wang H M, 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*, 2016, 74: 204-218.

[8] Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. In *Proc. the 36th Int. Conf. Software Engineering*, May 31-June 7, 2014, pp.356-366.

[9] Yu Y, Yin G, Wang T, Yang C, Wang H M. Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, 2016, 59: 080104.

[10] Thongtanunam P, McIntosh S, Hassan A E, Iida H. Investigating code review practices in defective files: An empirical study of the QT system. In *Proc. the 12th Working Conf. Mining Software Repositories*, May 2015, pp.168-179.

[11] Storey M A, Singer L, Cleary B, Filho F F, Zagalsky A. The (r)evolution of social media in software engineering. In *Proc. the Future of Software Engineering*, May 31-June 7, 2014, pp.100-116.

[12] Zhu J X, Zhou M H, Mockus A. Effectiveness of code contribution: From patch-based to pull-request-based tools. In *Proc. the 24th ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, November 2016, pp.871-882.

[13] De Lima M L, Soares D M, Plastino A, Murta L. Developers assignment for analyzing pull requests. In *Proc. the 30th Annual ACM Symp. Applied Computing*, April 2015, pp.1567-1572.

[14] van der Veen E, Gousios G, Zaidman A. Automatically prioritizing pull requests. In *Proc. the 12th Working Conf. Mining Software Repositories*, May 2015, pp.357-361.

[15] Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. In *Proc. the 35th Int. Conf. Software Engineering*, May 2013, pp.712-721.

[16] Rigby P C, Bacchelli A, Gousios G, Mukadam M. A mixed methods approach to mining code review data: Examples and a study of multi-commit reviews and pull requests. In *The Art and Science of Analyzing Software Data*, Bird C, Menzies T, Zimmermann T (eds.), Morgan Kaufmann, 2015, pp.231-255.

[17] Vasilescu B, Yu Y, Wang H M, Devanbu P, Filkov V. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proc. the 10th Joint Meeting on Foundations of Software Engineering*, August 30-September 4, 2015, pp.805-816.

[18] Mcintosh S, Kamei Y, Adams B, Hassan A E. An empirical study of the impact of modern code review practices on software quality. *Empirical Software Engineering*, 2016, 21(5): 2146-2189.

[19] Fagan M. Design and code inspections to reduce errors in program development. In *Software Pioneers*, Broy M, Denert E (eds.), Springer-Verlag, 2002, pp.575-607.

[20] Aurum A, Petersson H, Wohlin C. State-of-the-art: Software inspections after 25 years. *Sofware: Testing Verification and Reliability*, 2002, 12(3): 133-154.

[21] Rigby P, Cleary B, Painchaud F, Storey M A, German D. Contemporary peer review in action: Lessons from open source development. *IEEE Software*, 2012, 29(6): 56-61.

[22] Rigby P C, Storey M A. Understanding broadcast based peer review on open source software projects. In *Proc. the 33rd Int. Conf. Software Engineering*, May 2011, pp.541-550.

[23] Baum T, Liskin O, Niklas K, Schneider K. Factors influencing code review processes in industry. In *Proc. the 24th ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, November 2016, pp.85-96.

[24] Mcintosh S, Kamei Y, Adams B, Hassan A E. The impact of code review coverage and code review participation on software quality: A case study of the QT, VTK, and ITK projects. In *Proc. the 11th Working Conf. Mining Software Repositories*, May 31-June 1, 2014, pp.192-201.

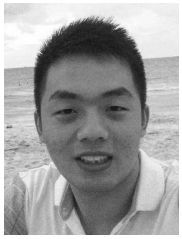
[25] Thongtanunam P, Mcintosh S, Hassan A E, Iida H. Review participation in modern code review. *Empirical Software Engineering*, 2016, 22(2): 768-817.

[26] Zhang Y, Wang H M, Yin G, Wang T, Yu Y. Social media in GitHub: The role of @-mention in assisting software development. *Science China Information Sciences*, 2017, 60: 032102.

[27] Baeza-Yates R A, Ribeiro-Neto B. Modern Information Retrieval: The Concepts and Technology Behind Search (2nd edition). Addison Wesley, 2011.

[28] Zhou Y, Tong Y X, Gu R H, Gall H. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, 2016, 28(3): 150-176.

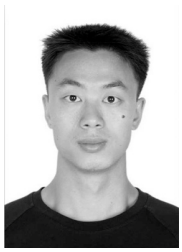
[29] Shah S K. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 2006, 52(7): 1000-1014.



Zhi-Xing Li is a Master student in the College of Computer, National University of Defense Technology, Changsha. His work interests include open source software engineering, data mining, and knowledge discovering in open source software.

Yue Yu is an assistant professor in the College of Computer, National University of Defense Technology, Changsha. He received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 2016. He visited University of California Davis supported by China Scholarship Council scholarship. His research findings have been published on MSR, FSE, IST, ICSME, etc. His current research interests include software engineering, spanning from mining software repositories and analyzing social coding networks.

Gang Yin is an associate professor in the College of Computer, National University of Defense Technology, Changsha. He received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 2006. He has published more than 60 research papers in international conferences and journals. His current research interests include distributed computing, information security, software engineering, and machine learning.



Tao Wang is an assistant professor in the College of Computer, National University of Defense Technology, Changsha. He received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 2015. His work interests include open source software engineering, machine learning, data mining, and knowledge discovering in open source software.

Huai-Min Wang is a professor in the College of Computer, National University of Defense Technology, Changsha. He received his Ph.D. degree in computer science from National University of Defense Technology, Changsha, in 1992. He has been awarded the "Chang Jiang Scholars Program" professor and the "Distinct Young Scholar", etc. He has published more than 100 research papers in peer-reviewed international conferences and journals. His current research interests include middleware, software agent, and trustworthy computing.