Glasgow Caledonian University

University for the Common Good

Code reviewer intelligent prediction in open source industrial software project

Liao, Zhifang; Zhang, Bolin; Huang, Xuechun; Yu, Song; Zhang, Yan

Published in:
Computer Modeling in Engineering & Sciences

DOI:
10.32604/cmes.2023.027466

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Link to publication in ResearchOnline

Citation for published version (Harvard):
Liao, Z, Zhang, B, Huang, X, Yu, S & Zhang, Y 2023, 'Code reviewer intelligent prediction in open source industrial software project', *Computer Modeling in Engineering & Sciences*, vol. 137, no. 1, pp. 687-704. https://doi.org/10.32604/cmes.2023.027466

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please view our takedown policy at https://edshare.gcu.ac.uk/id/eprint/5179 for details of how to contact us.

Download date: 13. Aug. 2023

Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2023.027466

ARTICLE

Code Reviewer Intelligent Prediction in Open Source Industrial Software Project

Zhifang Liao[1], Bolin Zhang[1], Xuechun Huang[1], Song Yu[1,*] and Yan Zhang[2]

[1]School of Computer Science and Engineering, Central South University, Changsha, 410083, China

[2]Department of Computing, School of Computing, Engineering and Built Environment, Glasgow Caledonian University, Glasgow, G4 0BA, UK

*Corresponding Author: Song Yu. Email: ys@csu.edu.cn

Received: 31 October 2022 Accepted: 22 December 2022

ABSTRACT

Currently, open-source software is gradually being integrated into industrial software, while industry protocols in industrial software are also gradually transferred to open-source community development. Industrial protocol standardization organizations are confronted with fragmented and numerous code PR (Pull Request) and informal proposals, and different workflows will lead to increased operating costs. The open-source community maintenance team needs software that is more intelligent to guide the identification and classification of these issues. To solve the above problems, this paper proposes a PR review prediction model based on multi-dimensional features. We extract 43 features of PR and divide them into five dimensions: contributor, reviewer, software project, PR, and social network of developers. The model integrates the above five-dimensional features, and a prediction model is built based on a Random Forest Classifier to predict the review results of PR. On the other hand, to improve the quality of rejected PRs, we focus on problems raised in the review process and review comments of similar PRs. We propose a PR revision recommendation model based on the PR review knowledge graph. Entity information and relationships between entities are extracted from text and code information of PRs, historical review comments, and related issues. PR revisions will be recommended to code contributors by graph-based similarity calculation. The experimental results illustrate that the above two models are effective and robust in PR review result prediction and PR revision recommendation.

KEYWORDS

Open source software; pull request; random forest; knowledge graph

1 Introduction

Software in open-source communities is gradually being integrated into complex industrial software systems [1], and it has become the new norm in the Industry 4.0 mode to accept open-source software and develop with it together. At the same time, some protocols and components in industrial systems are also gradually open-sourcing, to embrace the development of the community [2], promote open innovation and industrial synergy, and reduce the implementation threshold of multi-system interfacing in the industrial production chain [3]. However, the development form of open

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

source software is different from industrial software, especially in areas like development tools and organizational structure. The open-sourcing of industrial applications and protocols will increase the human cost of maintaining the project itself, and its standardization organization will face a large number of informal proposals, suggestions, and consultations.

In GitHub, an open-source social coding platform [4], suggestions and questions are submitted as Issues, proposals for changes to the software project are submitted as PRs (Pull Requests), and usually, a valid PR will fix the problem described in the Issue. However, the existing code PRs lack a clearly defined scoring mechanism. Code reviewers have to manually check code changes in PRs to make judgments. An automatic way to filter valuable PRs can save lots of time and effort for reviewers. Therefore, we try to study the existing manual PR classification and review process, and construct an automated PR classification and review prediction tool to help reviewers select more valuable PRs, and help contributors solve low-level errors. In this way, both reviewers and contributors can improve communication efficiency and promote the open-source development of industrial software.

To address these problems, this paper proposes a multi-dimensional feature-based PR review result prediction method MFPRPre (Multi-dimensional feature PR prediction). MFPRPre regards whether a PR will pass review as a dichotomous problem, and selects 43 features from five dimensions: contributor, reviewer, project, PR, and social network of developers. A Random Forest classifier is constructed to predict the result of the PR review. For the PRs that are not accepted, this paper also proposes a knowledge graph-based modification suggestion recommendation method (KGMORec). For the given PR, text and code information of PR, related issues, and review comments are analyzed to find the most similar PR entities in the knowledge graph. Then, KGMORec will recommend the most similar review comments to contributors, thus improving the passing rate of code review.

2 Related Work

The work related to the content of this paper focuses on two areas: the study of industrial software and open source software practices, and the study of PR revision related to open source software management practices.

The development practice and management experience of open source software is valuable for the development and open sourceization of industrial software. Therefore, many studies focus on the use of open source software in the development of industrial software. Linden et al. [1] found that industrial software is growing relying on open source software projects for development. This phenomenon implies that industrial software development requires knowledge of open source software cooperation and also requires middleware suppliers to update their products based on open source standards. Agerfalk et al. [5] proposed that lots of problems exist before open source projects being applied to industrial projects, such as intellectual property rights, development modes, and the skills required to participate in open source projects. Hunsen et al. [6] focused on the differences in the application of C preprocessors (CPP) in the open source and industrial domains. They confirmed that research on CPP can be effectively transferred from open source systems to industrial systems. Ebert [7] believed that free and open source software (FOSS) simplifies the complexity of software development. The development of industrial software will unavoidably use open source components as the basis of its development. Software suppliers provide the stability of commercial software by offering FOSS-based solutions to business users. Also, due to the FOSS-based project, this software supply activity is not monopolized by a specific software vendor.

The current researches on PR review prediction in GitHub focus on the textual description information of PRs. Marlow et al. [8] explored the main factors influencing the merging of PRs

by comparing data and analyzing various characteristics of successful and failed merged PRs. Soares et al. [9] found that the following factors have an impact on the merging of PRs: programming language, number of commits, files added, external developers and the first PR submitted by the contributor history. Ram et al. [10] conducted an empirical study and found that three main factors influenced the reviewability of PR: code changes, change descriptions, and commit history. Kim et al. [11] proposed a PR prioritization method-PRioritizer, which provides a prioritization method for reviewers facing multiple PRs, taking into account dynamic and static information of PRs. Jiang et al. [12] mainly considered modified code features, PR description text features, historical developer behavior features and project features, and combined these features to propose a CTCPPre method to predict the accepted PRs in GitHub. Studies on open source ecology have consistently shown that the results of PR audits are also related to social relationships [13]. Core project team members utilize social information when evaluating PRs [14], and regular developers can also generate impressions of the project through comments, showing different emotions and behaviors that form the potential personality of the user [8,14].

Current research works lack of content that is closely related to PR review comments in the open source community [4]. Some researchers have worked around duplicate detection of defect reports. Runeson et al. [15] was the first to address the problem of duplicate defect reports, he evaluated natural language processing methods to achieve duplicate detection and used Jaccard distance to calculate the similarity. Wang et al. [16] automatically detected duplicate reports by combining execution information and textual content in the defect report, reducing the cost of software development and maintenance. Nguyen et al. [17] modeled error reports as specific documents and resolved the submission of duplicate error reports by detecting similarities between error reports. Sun et al. [18] implemented a discriminative model to match similar defect reports and experimentally found that the model improved compared to NLP in three large defect repositories.

From related works we can find that in the task of predicting PR review results, researchers have disputed which features affect PR reviews. Also, there is a lack of focus on how to help contributors modify their PRs. A large number of studies have focused more on code defects themselves rather than PRs submitted by contributors. Our work focuses on the selection of PR features, PR review knowledge graph construction and PR revision recommendations to improve the quality of PRs in the open source community.

3 Approach

In this section, we will introduce the architecture for PR review result predication model and PR revision suggestion model.

3.1 Research Questions

To solve the above problems, this paper first proposes a PR review result prediction method called MFPRPre, which is based on a random forest classifier. To evaluate PR more comprehensively, we collected opinions from 56 people on the factors affecting the PR by questionnaires, with the opinions we selected 43 features from five dimensions. Secondly, for PRs that need to be modified, this paper designs a review comments recommendation method called KGMORec. The method combines domain knowledge mapping technology to effectively organize various entities such as textual and code information of PRs, historical review comments of PRs, and related Issues, to explore potential relationships among different knowledge entities and recommend review comments for defective PRs. This paper focuses on the following three questions:

Question 1: Do all the data dimension features affect the prediction of the prediction model? Which data dimension features are more important for the results?

Question 2: How is the performance of MFPRPre in predicting PR review results compared to existing models?

Question 3: How does KGMORec perform in recommending PR review comments compared to traditional recommender systems?

3.2 The Architecture for Models

The architecture for the two models is shown in Fig. 1. The architecture consists of three parts. The first part includes data collection and data procession, and it tries to collect the PR-related data. The second part includes features extraction and prediction model training (MFPRPre). If the prediction result is 'accept', the inspector will be prompted that the current PR is of high-quality. Otherwise, it is assumed that the PR still needs to be modified. The third part is KGMORec construction. We need to construct the PR knowledge graph and try to match the similar PRs in the knowledge graph by calculating the similarity of both code and text. Valuable review comments of these similar PRs are recommended to developers.

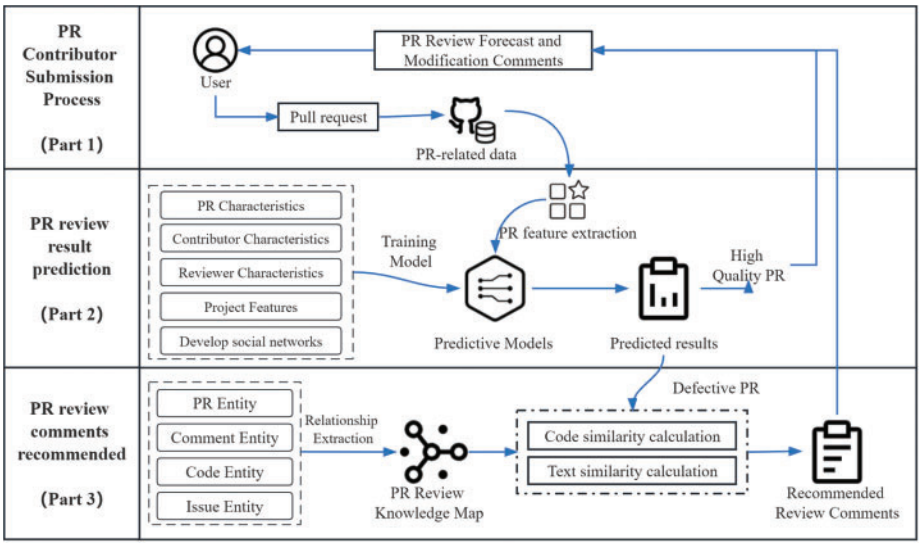

Figure 1: The model structure of MFPRPre and KGMORec

3.3 Prediction Model of PR Review Results

This section mainly describes the feature selection and construction process of the MFPRPre model in Fig. 1. Based on previous studies [10,19] and a survey of GitHub developers [20,21] and the survey of software engineers, we extract 43 features that may affect the review results of PR from 5 dimensions. A complete list of relevant features is presented in Table 1.

1. **Contributor feature extraction.** In this dimension, we mainly consider the user's community status and his or her activity in the project. In general, the community status of a contributor consists of his or her attention and the amount of code sharing, and the contributor's time and recent activity in the project also influence the reviewer's attitude. Based on existing studies [13,22], we select contributors' activity, contributor's historical PR status, and contributor's identity in the project as the features in the contributor dimension.

2. **Reviewer feature extraction.** In this dimension, we consider the impact of the project's popularity and activity on the review results. In the calculation of project popularity, we mainly consider the recent Star, Watch, and Fork counts of the project, and normalize the popularity. In the calculation of project activity, we will mainly consider the frequency of recent PRs and the PR response frequency of the project. Based on existing studies [8,23], we select part of the features of the reviewers such as the number of PR comments, the number of PR participants, and PR code tests. In addition, we add issue data (the number of associated Issues) as an important feature.
3. **Project feature extraction.** In this dimension, we consider the impact of PR descriptions and features in code changes on PR reviews, where PR descriptions help reviewers understand the purpose and characteristics of PRs, and the number and scope of code changes significantly affect the difficulty of the review and the review results. Based on existing studies, we select programming language and domain [24], project age, team size, and project popularity [14] as important project features. In order to better evaluate the popularity of projects, we incorporate the number of Star, Watch and Fork as projects' popularity features. In addition, we specifically add PR Waiting Time, PR Submission Count, and PR Acceptance Rate as project features to assess the overall work pace and style of the project.
4. **PR feature extraction.** During the code review process, the reviewers' attitudes change as they are discussed, and the PR will be approved or disapproved during multiple communications. Based on existing studies, we select the number of deleted lines of code, the number of new lines of code, and the number of modified files as code features of PR [9]. In addition, the similarity of PR description information [10] and PR text information are selected as PR text features. We believe that text similarity of PR can effectively express the personality of the reviewer and better highlight important functions of project.
5. **Developer social network feature extraction.** In this dimension, we mainly consider the activity characteristics of contributors in the social network. We rely on PR and Issue data mining to generate directed edges from reviewers to contributors to build a collaborative network [25]. We also calculate the social features of users in this collaborative network to measure the social distance of developers in the project [26]. The feature vector centrality is calculated as shown in Eq. (1). $DN(n_i)$ denotes the set of direct neighbors of node n_i, and n_i is the maximum eigenvalue of the adjacency matrix.

$$Ec(n_i) = \frac{1}{\lambda} \sum_{n_j \in DN(n_i)} Ec(n_j) \tag{1}$$

The construction process of the MFPRPre model consists of the following three main stages:

1. **Data collection and pre-processing:** Standard interface provided by GitHub is used to obtain popular open-source software projects. The collected data includes the text description and code of PR, contributors, reviewers, Issues and other related information. After that, a series of data pre-processing steps are performed, such as data cleaning, normalization, data transformation, feature selection and extraction.
2. **Multi-dimensional feature extraction:** We extract feature vectors of projects and their PRs in 5 dimensions.
3. **The construction of PR review result prediction model:** Features of 5 dimensions are used as inputs to the Random Forest Classifier to predict the PR review results.

Table 1: MFPRPre selected PR features

Feature	Description	Feature	Description
ctr_followers	Number of followers of a PR contributor	ctr_role	Whether the contributor is a core member
ctr_repo_num	Number of public repositories for a PR contributor	ctr_issue_num	Number of Issues submitted by a contributor
ctr_pr_num	Number of PRs submitted by a contributor historically	ctr_github_age	Length of time in GitHub
ctr_accept_rate	Acceptance Rate of PRs submitted by a contributor	ctr_first_step	Whether a contributor submits a PR for the first time
rvr_commit_num	Number of comments in PR	ctr_active	Whether the contributor has participated in PRs in recent three months
rvr_issues_fix	Whether a PR fixs Issue	net_connection	Number of contributors interacting with core members
rvr_issue_num	Number of comments of related Issues	net_connection_rate	Percentage of contributors interacting with core members
rvr_participant	Number of reviewers	net_centrality_measures	Centrality Metric
rvr_reviewer_rate	Percentage of reviewers responding to comments	net_closeness_centrality	Proximity Centrality
rvr_first_time	Time of the first review	net_betweenness_centrality	Intermediacy centrality
rvr_test_code	Does PR pass the code test	net_eigenvector_centrality	Feature vector metric
repo_age	Age of project	pr_contain_body	Whether a PR contains body content
repo_dev_num	Number of project developers	pr_changed_lines	Number of changed rows
repo_star	Number of Stars	pr_files_add	Number of added files
repo_watch	Number of Watchs	pr_files_deleted	Number of deleted files
repo_fork	Number of Forks	pr_files_modified	Number of modified files
repo_popularity	Project Popularity	pr_files_changed	Number of changed files
repo_language	Programming languages used in the project	pr_src_files	Number of source code files involved in the PR
repo_field	The field to which the project belongs	pr_doc_files	Number of documents involved in PR
repo_open_time	Average time for PR merge	pr_other_files	Number of non-structural resources involved in PR
repo_commit	Average number of submission	pr_text_sim	Similarity of text similarity of PR within 3 months
repo_accept_rate	Acceptance rate of PR		

3.4 Recommendation Model for PR Revision Suggestions

The model proposed in this paper can be used to recommend revision suggestions for PRs that fail to pass the review. Three modules of the KGMORec are: (1) the knowledge graph for PR revision; (2) the similarity calculation module, (3) and the recommendation module, as shown in Fig. 2.

Complex entity relationship information involved in PR revision process is effectively organized in the KG. The similarity calculation module is used to calculate the text similarity and code similarity between a failed PR and PR entities in KG, respectively. Finally, suitable review comments of Top-k similar PRs are listed as recommendations in the recommendation module.

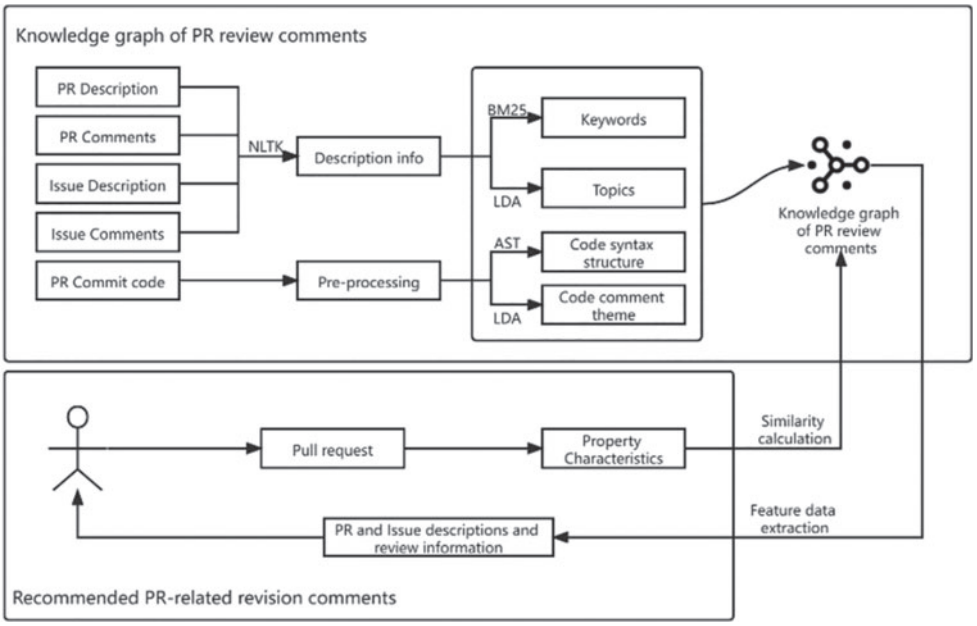

Figure 2: The overall structure of KGMORec

3.4.1 Definition of Ontology

The PR mechanism provided by GitHub encourages developers to submit PRs for Issues in open-source projects. After reviewing, results and specific revision comments of PRs will be given by reviewers. Developers can further modify and resubmit PRs according to revision comments. Core entities involved in the PR mechanism ontology include developers, reviewers, Issues, PRs and related comments. Source code contains a large number of code entities, which are interconnected based on syntax rules of different programming languages.

This paper integrates these two ontologies based on the impact of submitted PRs on source code, and the complete ontology is shown in Fig. 3.

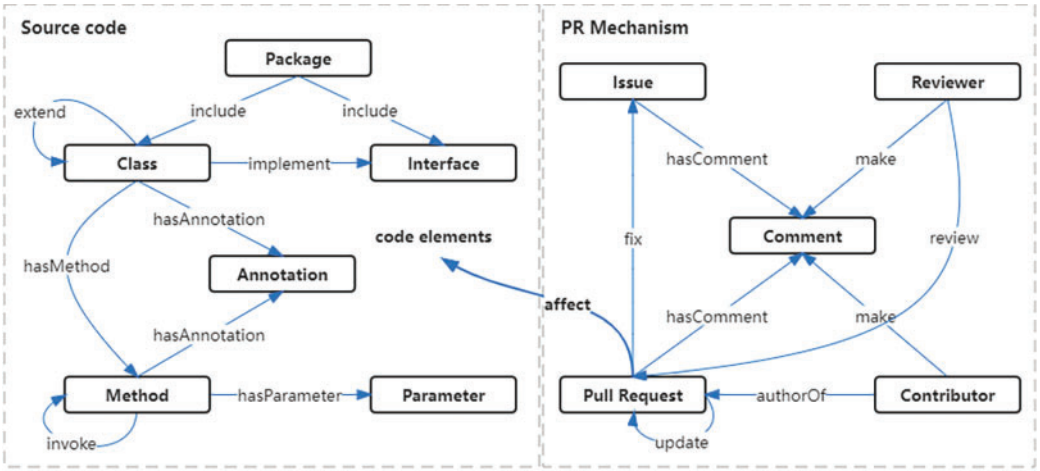

Figure 3: The ontology of knowledge graph for PR revision

3.4.2 Entity Extraction

Entities need to be extracted from both the source code and the PR mechanism. Take entity extraction of source code as an example, we divide knowledge of source code into three categories in this paper: (1) Code structure knowledge, such as class, interface, methods, etc. (2) Code description knowledge like code comments, (3) Code update knowledge, referring to the change history of source code.

In Table 2, we list entities involved in the source code ontology with their attributes. Specifically, JavaParse is used to parse Java code and generate abstract syntax trees, and we get entities and their attributes by traversing nodes of the tree. Core entities and their attributes involved in the PR mechanism are extracted in a similar way.

Table 2: MFPRPre selected PR features

Source code knowledge entities and attribute
Entity set of source code = {package, class, interface, variable, method, comment}
Attribute set of package = {package name}
Attribute set of class = {class name, modifier, class template, inherited parent class, package}
Attribute set of interface = {interface name, fully qualified name, template of the interface, parent interface, package}
Attribute set of variable = {variable name, class to which it belongs, modifier, type of variable}
Attribute set of method = {method name, class to which it belongs, modifier, parameter list, type of return value}
Attribute set of comment = {comment description}

3.4.3 Establishment of Relations between Entities

Some discrete knowledge entities can be obtained after entity extraction. However, it is still necessary to further establish the relationship between discrete knowledge entities to form a connected network integrating PR mechanism and source code knowledge system to formally express the PR revision scenario. This paper establishes relationships of code entities, relationships of PR mechanism entities, and relationships between PRs and code entities. Taking relationships of PR mechanism entities as an example, we carry out relationship extraction centered on 3 core entities, PR, Issue and Comment. The two types of relationships extracted centered on PR are shown in Table 3, which are Author relationship and Update relationship.

Table 3: Relationship table of PR mechanism entities

Relationship set of PR mechanism entities
Relationship set = {Author, Update}
Author = {Developer, Pull Request}
Update = {Pull Request, resubmited Pull Request}

3.4.4 The Similarity Calculation Module

We calculate the explicit text similarity and topic similarity of PRs based on the BM25 algorithm and the LDA model, and combine the two as the final text similarity $SimText(pr_i, pr_j)$.

We consider semantic structure of code as well as the code comments when calculating the code similarity. The semantic structure similarity is calculated based on AST, and the node representation is first generated by the feature vector method. Then, the edit distance between feature vectors is calculated using a locally sensitive hashing algorithm, as shown in Eq. (2). T_1 and T_2 denote the vectors of AST of pr_i and pr_j. $SimText(T_1, T_2)$ denotes the number of similar nodes in T_1 and T_2. $T_1 \times Size + T_2 \times Size$ is the number of all nodes in two ASTs.

$$SimCodeNode(pr_i, pr_j) = \frac{2 * Sim(T_1, T_2)}{T_1 * Size + T_2 * Size} \tag{2}$$

For code comments, we use the LDA model to generate topic probability distribution and then calculate the topic similarity $Comment(pr_i, pr_j)$ of code comments. As shown in Eq. (3), the final code similarity is the weighted sum of the two, where γ and δ are weighting factors.

$$SimCodeNode(pr_i, pr_j) = \gamma \times SimCodeNode(pr_i, pr_j) + \delta \times Comment(pr_i, pr_j) \tag{3}$$

3.4.5 The Recommendation Module

As shown in Eq. (4), the final similarity score of pr_i and pr_j is obtained by fusing the text similarity and the code similarity with a certain factor α. The Top-k PR entities in the knowledge graph are selected based on similarity ranking, suitable review comments of which will be organized as the final recommending list of the KGMORec model.

$$Similarity(pr_1, pr_2) = \alpha \times SimText(pr_1, pr_2) + (1 - \alpha) \times SimCode(pr_1, pr_2) \tag{4}$$

4 Experiment and Analysis

In order to verify the performance of the MFPRPre prediction model and KGMORec recommendation model proposed in this paper, a large amount of open source project data was collected from the GitHub platform, and an experimental environment was configured for training and testing the model.

4.1 Description of Data Sets and Indicators

We have selected the 20 most popular GitHub open source projects, and they all use PR as the main development method.

(1) Dataset for MFPRPre. We collected information related to project PRs through the GitHub API to form a dataset for the MFPRPre prediction model study. The dataset contains 20 large software projects and 216,920 PRs, of which 128,326 were accepted. Nine projects in the dataset have an acceptance rate of more than 0.7 and 5 projects have an acceptance rate of less than 0.5. Among the 20 research projects, reactiveX/RxJava and elastic/elasticsearch projects have the highest acceptance rate of 0.85, indicating that these two projects are more likely to accept PRs, while the acceptance rate of nodejs/node project is only 0.07, indicating that the project reviewers rarely accept external contributions. The basic statistics of the projects in the dataset are shown in Table 4.

(2) Dataset for KGMORec. In order to keep the consistency of our experiment, we use the same dataset as MFPRPre. Based on the possibility that the differences in syntax structure of

different program languages may have an impact on data processing and final recommendation results, we select 6 projects using Java from the whole dataset. Statistics of the selected Java projects are shown in Table 5. PR and issue stand for the number of PRs and Issues submitted to the project by contributors, respectively. PR Comment stands for the number of comments that developers and reviewers discuss on the submitted PR. Revised PR stands for the number of PRs that was not accepted the first time, was modified and then resubmitted. Accepted PR stands for the number of PRs that were accepted after the second modification. Obviously, the modification of PRs is frequent development behavior.

Table 4: Statistic of MFPRPre dataset

Projects	PR	AC	PR pass rate
Scala/scala	8775	6756	0.77
Facebook/react	9381	6640	0.71
Tensorflow/tensorflow	13919	9566	0.69
Twbs/bootstrap	10744	5651	0.53
Ohmyzsh/ohmyzsh	5025	2220	0.44
Cocos2d/cocos2d-x	15559	12320	0.63
Flutter/flutter	19517	14388	0.74
Nodejs/node	20643	2282	0.07
Alibaba/nacos	1518	1133	0.75
Ant-design/ant-design	6504	5180	0.80
Angular/angular.js	7829	770	0.10
Keras-team/keras	3822	2402	0.63
Mockito/mockito	1052	810	0.77
Bitcoin/bitcoin	12706	8635	0.68
Pytorch/pytorch	21932	4079	0.19
Square/okhttp	2832	2347	0.83
Apache/dubbo	3150	1944	0.62
Elastic/elasticsearch	43825	37179	0.85
Reactivex/RxJava	3553	3020	0.85
Apache/spark	5100	1376	0.27
Total	217386	128698	0.59

Table 5: Statistic of selected Java projects

Project	PR	Issue	PR comment	Revised PR	Accepted PR
Alibaba/nacos	1518	3440	5160	349	216
Mockito/mockito	1052	1177	2945	252	163
Square/okhttp	2832	3354	5947	1099	267
Apache/dubbo	3150	936	7135	1320	570

(Continued)

Table 5 (continued)

Project	PR	Issue	PR comment	Revised PR	Accepted PR
Elastic/elasticsearch	43825	26599	77265	14812	9539
Reactivex/rxJava	3553	3022	6920	906	543
Total	55930	38528	105373	18738	11298

4.2 Multidimensional Feature Validity Analysis

To explore whether all data dimension features affect the model prediction, this paper constructs prediction models based on five single feature dimensions: contributor features, reviewer features, software project features, PR features, and developer social network. The accuracy and AUC values of the prediction results based on single dimensional features are presented in Table 6. We find that the combined set of five dimensions outperforms the contributor-only features, with accuracy rates of 0.02, 0.02, 0.06, 0.02, and 0.07 higher than the five single-dimension features, respectively.

Table 6: Accuracy and AUC of prediction results based on single features

Project	Contributors	Projects	PR	Reviewer	Social networking	All features
Scala	0.92/0.81	0.89/0.81	0.88/0.80	0.90/0.83	0.80/0.78	0.92/0.86
React	0.80/0.71	0.80/0.70	0.78/0.69	0.80/0.70	0.74/0.65	0.83/0.71
Tensorflow	0.71/0.72	0.72/0.71	0.70/0.71	0.71/0.74	0.66/0.70	0.73/0.77
Bootstrap	0.72/0.70	0.72/0.70	0.75/0.69	0.75/0.70	0.69/0.71	0.75/0.74
Ohmyzsh	0.92/0.85	0.91/0.81	0.90/0.80	0.90/0.84	0.90/0.78	0.92/0.85
Cocos2d-x	0.78/0.76	0.80/0.71	0.77/0.70	0.80/0.71	0.78/0.67	0.81/0.76
Flutter	0.76/0.62	0.76/0.60	0.75/0.59	0.74/0.65	0.77/0.60	0.79/0.69
Node	0.86/0.66	0.86/0.66	0.84/0.60	0.84/0.66	0.82/0.59	0.88/0.68
Alibaba/nacos	0.77/0.66	0.74/0.74	0.77/0.69	0.74/0.71	0.71/0.70	0.80/0.75
Ant-design	0.88/0.75	0.88/0.75	0.81/0.73	0.88/0.75	0.80/0.70	0.89/0.76
Angular.js	0.92/0.80	0.92/0.82	0.90/0.80	0.93/0.84	0.93/0.78	0.93/0.84
Keras	0.75/0.72	0.74/0.76	0.68/0.77	0.72/0.77	0.73/0.77	0.76/0.78
Mockito	0.80/0.80	0.81/0.86	0.70/0.80	0.82/0.81	0.78/0.80	0.83/0.86
Bitcoin	0.71/0.78	0.78/0.77	0.63/0.72	0.77/0.77	0.76/0.74	0.80/0.79
Pytorch	0.85/0.80	0.90/0.78	0.85/0.77	0.89/0.79	0.85/0.78	0.91/0.81
Okhttp	0.86/0.92	0.89/0.91	0.80/0.89	0.90/0.90	0.78/0.89	0.90/0.93
Dubbo	0.80/0.70	0.78/0.72	0.77/0.68	0.78/0.71	0.71/0.71	0.81/0.72
Elasticsearch	0.91/0.78	0.90/0.79	0.80/0.69	0.87/0.78	0.77/0.78	0.91/0.80
RxJava	0.80/0.82	0.77/0.84	0.72/0.78	0.76/0.80	0.70/0.82	0.80/0.86
Spark	0.78/0.71	0.85/0.66	0.81/0.66	0.84/0.70	0.71/0.68	0.85/0.71
Average	0.82/0.75	0.82/0.76	0.78/0.73	0.82/0.76	0.77/0.73	0.84/0.78

At the same time, we believe that selecting features of all five dimensions is more beneficial to obtain accurate prediction results, so we designed a set of experiments in which we cyclically remove one dimension of data and train. Predict in the selected code repository. As it can be seen from Table 7, the results of the experiments in which all five dimensions are selected are better than the experiments in which some dimensions are missing in terms of accuracy and AUC, so it shows that all the features selected by MFPRPre have a positive effect on the results.

Table 7: Accuracy and AUC of prediction results based on part of features

Project	Del contributors		Del projects		Del PR		Del reviewer		Del SNS		All features	
	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC	ACC	AUC
Scala	0.97	0.82	0.9	0.82	0.88	0.79	0.93	0.86	0.7	0.75	0.92	0.86
React	0.82	0.74	0.82	0.71	0.77	0.69	0.82	0.71	0.68	0.6	0.83	0.71
Tensorflow	0.72	0.72	0.74	0.7	0.7	0.7	0.72	0.77	0.61	0.68	0.73	0.77
Bootstrap	0.71	0.7	0.71	0.7	0.78	0.68	0.78	0.7	0.64	0.72	0.75	0.74
Ohmyzsh	0.94	0.89	0.92	0.8	0.89	0.78	0.89	0.87	0.89	0.74	0.92	0.85
Cocos2d-x	0.77	0.82	0.82	0.71	0.75	0.69	0.82	0.71	0.77	0.62	0.81	0.76
Flutter	0.76	0.63	0.76	0.58	0.74	0.56	0.72	0.7	0.79	0.58	0.79	0.69
Node	0.88	0.69	0.88	0.69	0.84	0.56	0.84	0.69	0.79	0.53	0.88	0.68
Alibaba/nacos	0.8	0.61	0.73	0.79	0.8	0.68	0.73	0.72	0.66	0.7	0.8	0.75
Ant-design	0.92	0.77	0.92	0.77	0.76	0.72	0.92	0.77	0.74	0.65	0.89	0.76
Angular.js	0.92	0.79	0.92	0.84	0.88	0.79	0.94	0.88	0.94	0.75	0.93	0.84
Keras	0.78	0.67	0.76	0.76	0.63	0.78	0.72	0.78	0.74	0.78	0.76	0.78
Mockito	0.82	0.78	0.85	0.92	0.6	0.78	0.87	0.81	0.78	0.78	0.83	0.86
Bitcoin	0.68	0.81	0.84	0.79	0.51	0.67	0.82	0.79	0.8	0.72	0.8	0.79
Pytorch	0.83	0.82	0.94	0.77	0.83	0.75	0.92	0.8	0.83	0.78	0.91	0.81
Okhttp	0.88	0.94	0.94	0.92	0.74	0.88	0.97	0.9	0.7	0.88	0.9	0.93
Dubbo	0.84	0.7	0.8	0.74	0.77	0.65	0.8	0.72	0.64	0.72	0.81	0.72
Elasticsearch	0.98	0.8	0.96	0.82	0.74	0.6	0.89	0.8	0.67	0.8	0.91	0.8
RxJava	0.86	0.83	0.8	0.87	0.68	0.74	0.77	0.79	0.64	0.83	0.8	0.86
Spark	0.76	0.74	0.91	0.63	0.83	0.63	0.89	0.72	0.6	0.68	0.85	0.71
Average	0.84	0.76	0.84	0.78	0.75	0.71	0.84	0.78	0.73	0.71	0.84	0.78

4.3 MFPRPre Prediction Model Analysis

To obtain the best prediction models, we experimented with five different machine learning models in a selection of 20 Github projects. A more uniform performance fluctuation in accuracy occurred for all classifiers across projects, but the random forest model always had a significant advantage. In Table 8, by calculating the mean values of the performance indicators, we can find that the accuracy of the random forest model is 0.85. The average accuracies of SVM, Decision Tree, Naive Bayesian and Logistic Regression were 0.81, 0.77, 0.73, 0.71, respectively. In the AUC performance index, the average AUC of random forest is 0.78, and the other models are 0.77, 0.74, 0.70, and 0.69 in that order. We can find that the prediction model based on random forest has more advantages in various indexes over other machine learning algorithm models through comparison experiments, so the MFPRPre model uses the random forest method as the base model.

Table 8: Performance of five model classifications

	Precision	Accuracy	Recall	F-score	AUC
Random	0.85	0.85	0.84	0.84	0.78
SVM	0.81	0.81	0.82	0.82	0.77
Decision	0.77	0.79	0.78	0.79	0.74
Bayesian	0.73	0.75	0.73	0.74	0.70
Logistic	0.71	0.71	0.7	0.71	0.69

In order to compare the performance of MFPRPre prediction model with other prediction models, two models, PRioritizer [27] and CTCPPre [12], are selected for comparison experiments. The PRioritizer model builds prediction models based on PR developer information, PR project information and PR content information, and the CTCPPre model mainly considers the code in PR modification features and PR text description features. Fig. 4 compares the data of MFPRPre, Prioritizer and CTCPPre in terms of Accuracy, AUC value, Precision, Recall and F1-score. The comparison revealed that CTCPPre outperformed the Prioritizer model, while our MFPRPre model outperformed CTCPPre by 0.01, 0.02, 0.01, 0.03, 0.03 on average. The experiments showed that the MFPRPre model was better than Prioritizer and CTCPPre in PR outcome prediction.

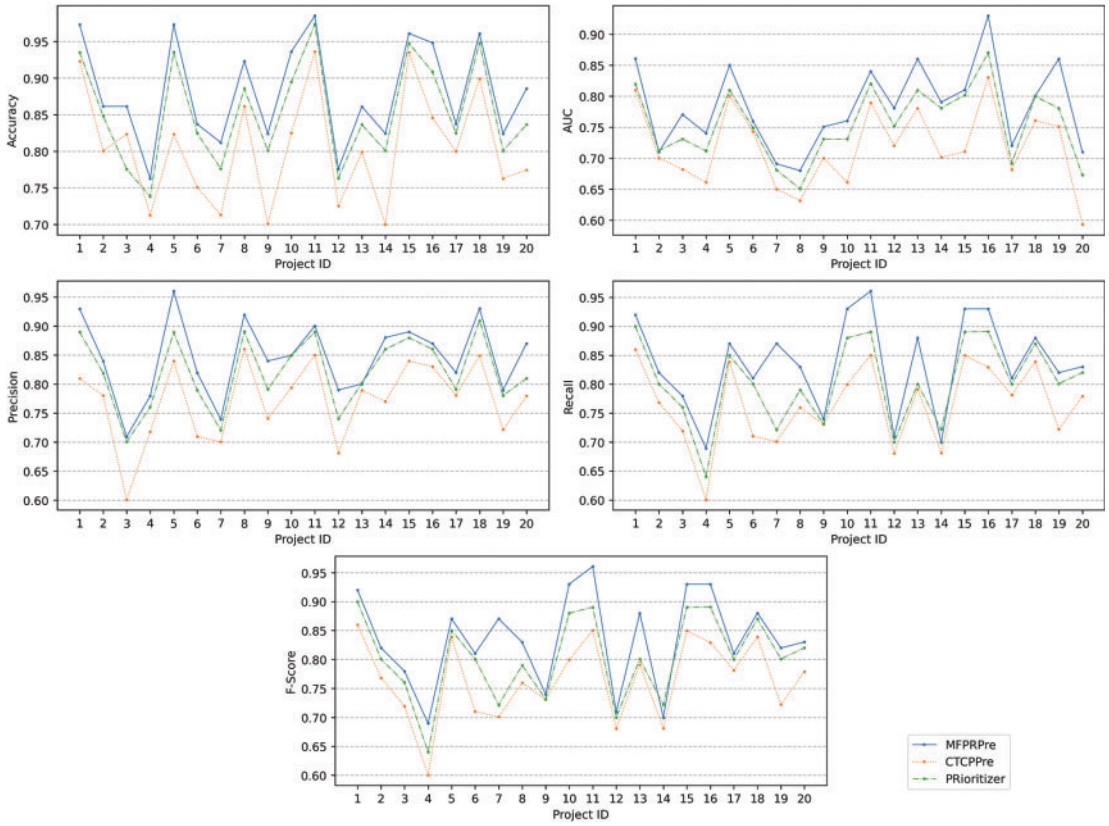

Figure 4: MFPRPre and Prioritizer and CTCPPre performance evaluation curves

4.4 KGMORec Prediction Model Analysis

Based on the dataset consisting of 6 Java projects, we construct the PR revision knowledge graph. Tables 9 and 10 list the types and corresponding number of entities in the knowledge graph. A total of 11 types, 1,107,305 entities, 14 relationships, and 345,445 edges are extracted.

Table 9: Type and number of entities

Type	Number	Type	Number	Type	Number
Class	88920	Method	177602	Interface	54738
Parameter	193839	Annotation	159302	Package	87321
PR	55930	Issue	38528	Commit	88928
Contributor	55930	Reviewer	106267		

Table 10: Type and number of relationships

Type	Between	Number
AuthorOf	(Contributor, PR)	55930
Review	(Reviewer, PR)	89271
Make	(Contributor, Comment) or (Reviewer, Comment)	30052
HasComment	(Issue-Comment) or (PR-Comment)	27920
Update	(PR, PR)	8903
Fix	(Issue, PR)	23798
Affect	(PR, code elements)	12148
Extend	(Class, Class)	13928
Implement	(Class, Interface)	12819
Invoke	(Method, Method)	15988
Include	(Package, Class) or (Package, Interface)	10392
HasParameter	(Method, Parameter)	14739
HasMethod	(Class, Method)	16647
HasAnnotation	(Method, Parameter)	12910

The key parameters of this experiment are the fusion factor α of text similarity and code similarity of the similarity calculation module, and the parameter K recommended by Top-K [28,29]. The value of the fusion factor α ranges from the interval [0,1], with a taken interval of 0.2. When $\alpha = 0$, the similarity calculation is based entirely on text similarity; when $\alpha = 1$, the similarity calculation is based entirely on code similarity. Fig. 5 gives a partial example of the knowledge graph that we constructed.

With different values of the fusion factor α, the experiments of PR modification recommendation for Top-5, Top-10, Top-15, Top-20 and Top-25 were conducted in this paper in turn, and the experimental results are shown in Fig. 6. From the information in the figures, we can see that the accuracy and recall rate reach the peak when the fusion factor $\alpha = 0.6$; and the best recommendation effect of the model is achieved when the K value of Top-K is 15.

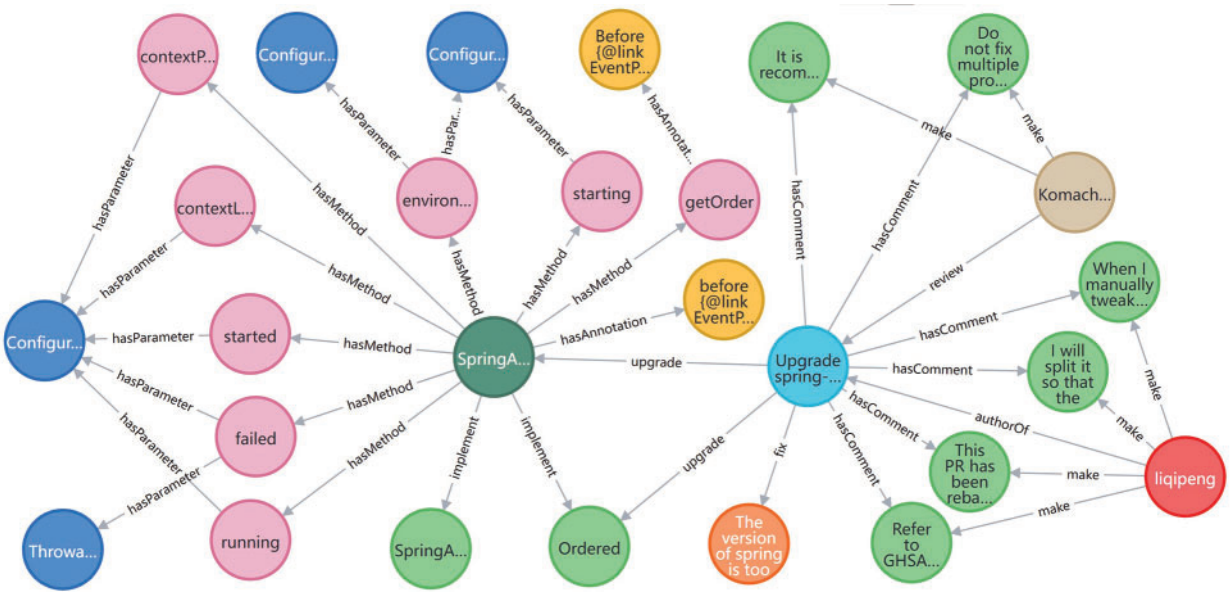

Figure 5: Example of KGMORec mapping structure

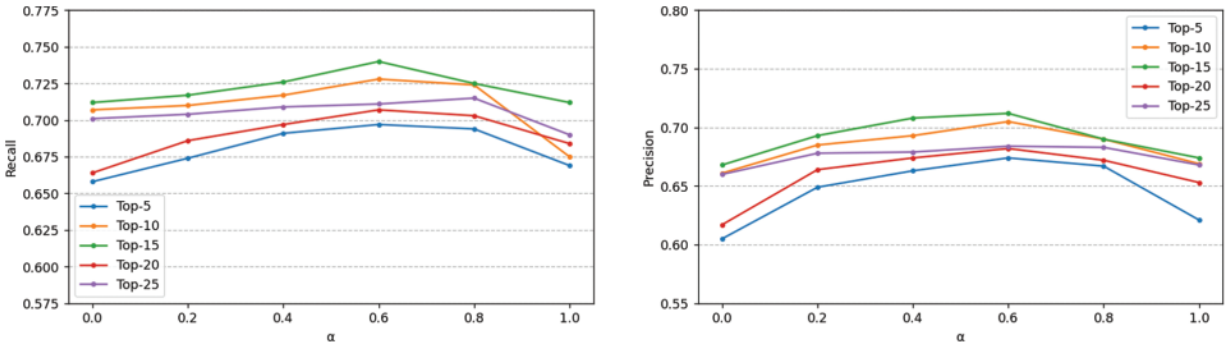

Figure 6: The relationship between recall and precision and factor

In order to verify the effectiveness of KGMORec method, the traditional collaborative filtering recommendation algorithm is selected for comparison in this paper, and the experimental results of the two methods are shown in Table 11. The experimental results demonstrate that the performance of the KGMORec method proposed in this paper on the Java dataset is significantly higher than that of the traditional collaborative filtering recommendation method, with significant improvements in accuracy, recall and F1-score [30].

Table 11: Hit rate of Top-K experiment

	Top-5	Top-10	Top-15	Top-20	Top-25
Item-CF	0.649	0.693	0.708	0.679	0.674
KGMORec	0.674	0.705	0.712	0.682	0.684

5 Conclusion

In this paper, we first investigate the process of filtering high-quality PR by open-source project reviewers and propose a PR review result prediction method, MFPRPre. The method was developed by selecting 43 valid features from the five dimensions of activity that received more attention in the review activity. We experimented with two independent experiments on dimensional validity to show that all selected features are beneficial to get better prediction results. Also, we experimentally compared multiple data mining classification algorithms to select the most appropriate classifier. With the help of the experimental data, we selected the random forest classifier as the basis of MFPRPre. In the baseline comparison experiments, the selection of multiple features and the random forest classifier enable us to achieve better performance advantages.

In the second model of this paper, we specifically focus on PR that is not approved during PR review activities, tracking the recommitting activities of this PR after the review. Through our research, this paper proposes a PR revision recommendation model based on the PR revision knowledge graph, KGMORec. This model mines the correlation between multiple PR knowledge entities and constructs a PR review knowledge graph. And then, it recommends PR revisions to contributors by graph-based similarity calculations. By comparing with traditional recommendation models, we showed that KGMORec has better performance in recommendation activities and it better exploits the potential relationships between knowledge entities to provide interpretable recommendation results.

Knowledge mapping technology is the key technology of the model described in this paper, which helps us to mine the data in PR review activities. However, the definition of entities and relationships in the knowledge graph proposed in this paper still differs from the objective world. Also, there are still many shortcomings to be improved in terms of the method of entity identification. More in-depth research and exploration are needed to obtain better results.

Acknowledgement: The authors wish to express their appreciation to the Central South University and the reviewers for their helpful suggestions which greatly improved the presentation of this paper.

Funding Statement: The authors thank the financial support of National Social Science Fund (NSSF) under Grant (No. 22BTQ033).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. van der Linden, F., Lundell, B., Marttiin, P. (2009). Commodification of industrial software: A case for open source. *IEEE Software, 26(4),* 77–83. https://doi.org/10.1109/MS.2009.88
2. Palm, F., Grüner, S., Pfrommer, J., Graube, M., Urbas, L. (2015). Open source as enabler for OPC UA in industrial automation. *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–6. Luxembourg, Luxembourg. https://doi.org/10.1109/ETFA.2015.7301562
3. Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O. et al. (2016). Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering, 21(2),* 449–482. https://doi.org/10.1007/s10664-015-9360-1
4. Liao, Z., Zhao, B., Liu, S., Jin, H., He, D. et al. (2019). A prediction model of the project life-span in open source software ecosystem. *Mobile Networks and Applications, 24(4),* 1382–1391. https://doi.org/10.1007/s11036-018-0993-3

5. Agerfalk, P. J., Deverell, A., Fitzgerald, B., Morgan, L. (2005). Assessing the role of open source software in the European secondary software sector: A voice from industry. *1st International Conference on Open Source Software*, Genoa, Italy.
6. Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O. et al. (2016). Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering, 21(2),* 449–482. https://doi.org/10.1007/s10664-015-9360-1
7. Ebert, C. (2008). Open source software in industry. *IEEE Software, 25(3),* 52–53. https://doi.org/10.1109/MS.2008.67
8. Marlow, J., Dabbish, L., Herbsleb, J. (2013). Impression formation in online peer production: Activity traces and personal profiles in Github. *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, New York, NY, USA, Association for Computing Machinery.
9. Soares, D. M., de Lima Júnior, M. L., Murta, L., Plastino, A. (2015). Acceptance factors of pull requests in open-source projects. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, New York, NY, USA, Association for Computing Machinery.
10. Ram, A., Sawant, A. A., Castelluccio, M., Bacchelli, A. (2018). What makes a code change easier to review: An empirical investigation on code change reviewability. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, Association for Computing Machinery.
11. Kim, D., Nam, J., Song, J., Kim, S. (2013). Automatic patch generation learned from human-written patches. *2013 35th International Conference on Software Engineering (ICSE)*, pp. 802–811. San Francisco, CA, USA. https://doi.org/10.1109/ICSE.2013.6606626
12. Jiang, J., Zheng, J. T, Yang, Y., Zhang, L. (2020). CTCPPre: A prediction method for accepted pull requests in GitHub. *Journal of Central South University, 27(2),* 449–468. https://doi.org/10.1007/s11771-020-4308-z
13. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, New York, NY, USA, Association for Computing Machinery.
14. Tsay, J., Dabbish, L., Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in GitHub. *Proceedings of the 36th International Conference on Software Engineering*, New York, NY, USA, Association for Computing Machinery.
15. Runeson, P., Alexandersson, M., Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. *29th International Conference on Software Engineering (ICSE'07)*, pp. 499–510. Minneapolis, MN, USA. https://doi.org/10.1109/ICSE.2007.32
16. Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. *Proceedings of the 30th International Conference on Software Engineering*, New York, NY, USA, Association for Computing Machinery.
17. Nguyen, A. T., Nguyen, T. T., Nguyen, T. N., Lo, D., Sun, C. (2012). Duplicate bug report detection with a combination of information retrieval and topic modeling. *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 70–79. Essen, Germany. https://doi.org/10.1145/2351676.2351687
18. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S. C. (2010). A discriminative model approach for accurate duplicate bug report retrieval. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, vol. 1. New York, NY, USA, Association for Computing Machinery.
19. Gousios, G., Pinzger, M., van Deursen, A. (2014). An exploratory study of the pull-based software development model. *Proceedings of the 36th International Conference on Software Engineering*. New York, NY, USA, Association for Computing Machinery.
20. Gousios, G., Zaidman, A., Storey, M. A., van Deursen, A. (2015). Work practices and challenges in pull-based development: The integrator's perspective. *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, pp. 358–368. Florence, Italy. https://doi.org/10.1109/ICSE.2015.55

21. Gousios, G., Storey, M. A., Bacchelli, A. (2016). Work practices and challenges in pull-based development: The contributor's perspective. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 285–296. Austin, TX, USA. https://doi.org/10.1145/2884781.2884826
22. Ford, D., Behroozi, M., Serebrenik, A., Parnin, C. (2019). Beyond the code itself: How programmers really look at pull requests. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, pp. 51–60. Montreal, QC, Canada. https://doi.org/10.1109/ICSE-SEIS.2019.00014
23. Lenarduzzi, V., Nikkola, V., Saarimäki, N., Taibi, D. (2021). Does code quality affect pull request acceptance? An empirical study. *Journal of Systems and Software, 171(1),* 110806. https://doi.org/10.1016/j.jss.2020.110806
24. Rahman, M. M., Roy, C. K. (2014). An insight into the pull requests of Github. *Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, NY, USA, Association for Computing Machinery.
25. Zhou, M., Mockus, A. (2015). Who will stay in the floss community? Modeling participant's initial behavior. *IEEE Transactions on Software Engineering, 41(1),* 82–99. https://doi.org/10.1109/TSE.2014.2349496
26. Zanetti, M. S., Scholtes, I., Tessone, C. J., Schweitzer, F. (2013). Categorizing bugs with social networks: A case study on four open source software communities. *2013 35th International Conference on Software Engineering (ICSE)*, pp. 1032–1041. San Francisco, CA, USA. https://doi.org/10.1109/ICSE.2013.6606653
27. van der Veen, E., Gousios, G., Zaidman, A. (2015). Automatically prioritizing pull requests. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 357–361. Florence, Italy. https://doi.org/10.1109/MSR.2015.40
28. Yang, P., Luo, X., Sun, J. (2022). A simple but effective method for balancing detection and re-identification in multi-object tracking. *IEEE Transactions on Multimedia, 2022,* 1–13. https://doi.org/10.1109/TMM.2022.3222614
29. Li, J., Luo, X., Ma, H., Zhao, W. (2022). A hybrid deep transfer learning model with kernel metric for COVID-19 pneumonia classification using chest CT images. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2022,* 1–12. https://doi.org/10.1109/TCBB.2022.3216661
30. Liao, Z., Song, T., Wang, Y., Fan, X., Zhang, Y. (2018). User personalized label set extraction algorithm based on lda and collaborative filtering in open source software community. *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pp. 1–5. Alsace, Colmar, France. https://doi.org/10.1109/CITS.2018.8440167