




Article

New Developer Metrics for Open Source Software Development Challenges: An Empirical Study of Project Recommendation Systems

Abdulkadir Şeker ^{1,2,*} , Banu Diri ³  and Halil Arslan ¹ 

¹ Department of Computer Engineering, Sivas Cumhuriyet University, 58140 Sivas, Turkey; harslan@cumhuriyet.edu.tr

² Renewable Energy Research Center, Sivas Cumhuriyet University, 58140 Sivas, Turkey

³ Department of Computer Engineering, Yıldız Technical University, 34349 İstanbul, Turkey; diri@yildiz.edu.tr

* Correspondence: aseker@cumhuriyet.edu.tr

Abstract: Software collaboration platforms where millions of developers from diverse locations can contribute to the common open source projects have recently become popular. On these platforms, various information is obtained from developer activities that can then be used as developer metrics to solve a variety of challenges. In this study, we proposed new developer metrics extracted from the issue, commit, and pull request activities of developers on GitHub. We created developer metrics from the individual activities and combined certain activities according to some common traits. To evaluate these metrics, we created an item-based project recommendation system. In order to validate this system, we calculated the similarity score using two methods and assessed top-*n* hit scores using two different approaches. The results for all scores with these methods indicated that the most successful metrics were *binary_issue_related*, *issue_commented*, *binary_pr_related*, and *issue_opened*. To verify our results, we compared our metrics with another metric generated from a very similar study and found that most of our metrics gave better scores than that metric. In conclusion, the issue feature is more crucial for GitHub compared with other features. Moreover, commenting activity in projects can be equally as valuable as code contributions. The most of binary metrics that were generated, regardless of the number of activities, also showed remarkable results. In this context, we presented improvable and noteworthy developer metrics that can be used for a wide range of open-source software development challenges, such as user characterization, project recommendation, and code review assignment.

Keywords: developer metric; open source; project recommendation system; GitHub; issue; pull request; commit

MSC: 68N30



Citation: Şeker, A.; Diri, B.; Arslan, H. New Developer Metrics for Open Source Software Development Challenges: An Empirical Study of Project Recommendation Systems. *Appl. Sci.* **2021**, *11*, 920. <https://doi.org/10.3390/app11030920>

Received: 28 November 2020

Accepted: 15 January 2021

Published: 20 January 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Thanks to the increasing capabilities of open source software (OSS) development tools, the number of open-source users and projects is growing each year. Existing software collaboration platforms include millions of developers with different characters and skill sets, as well as a wide variety of projects that offer solutions to various problems. GitHub, is the largest one among these platforms, that hosting more than 40 million repositories to which over 100 million developers have contributed. On this platform, several features are used to manage the distributed and open-source projects of which the most widely used are issues, commits, and pull requests (PRs). Activities related to these features—such as opening an issue, merging a PR, or commenting on a commit—can provide information about the developers and projects. The knowledge obtained from this information can then be used in the form of developer metrics to solve various software engineering challenges.

Some of the developer metrics used are the number of lines of code in a project, developer degrees of connection to one another, past experience, or common features (e.g., nationality, location, occupation, gender, previously used programming languages). These metrics offer solutions to different problems within OSS development and distributed coding, including automatic assignments (task, issue, bug, or reviewer) [1–3], project recommendation systems [4,5], and software defect detection [6].

However, on platforms with such a large amount of data, it is challenging for developers to find similar projects to their own, identify projects of interest, and reach projects to which they can contribute. As developers primarily use search engines or in-platform search menus to find projects, the constraints of text-based search [7] and challenges related to finding the correct keywords also lead them to miss some projects [8]. While various project recommendation systems are being developed to overcome this problem, projects must be rated by users if recommendation models are to work properly. In the same way that viewers give ratings to movies that they have watched, developers need to rate the projects in which they are interested. However, this does not currently occur on software collaboration platforms, meaning there is no labeled dataset for work on this problem. For this reason, several developer metrics that can be extracted from the activity or features of both developers and projects can be used to calculate the score that a user gives projects.

In this study, new developer metrics are presented to be used for a variety of open source distributed software development challenges. We developed a project recommendation system due to evaluate these metrics using data from GitHub with the aim of making recommendations to developers based on their GitHub activities. Moreover, to address the sparsity problem on GitHub, we selected a dataset with a high project-user ratio. Despite this handicap, we obtained remarkable results in comparison with a similar study [5]. We present the following research questions for this study:

- RQ1. Can we offer new evaluation methods for GitHub project recommendation problem?
- RQ2. Can we use the activities of GitHub users as a developer metric individually?
- RQ3. Can new metrics be obtained by combining similar properties or activities?
- RQ4. Is a user's amount of activity important in the context of developer metrics?

In light of these research questions, we organized this paper as follows. In the background section, we discuss the literature on previously proposed developer metrics that have been used for a wide range of OSS development challenges. In the following section, we describe our dataset, proposed metrics, and project recommendation model. We evaluate the metrics using different approaches and methods in the last section.

2. Background

The activities of developers in platforms such as GitHub provide collaboration, learning, and reputation management in the community [9]. These activities are the metadata of the platforms which directly correlated to reputation or performance of the developers [10]. Besides, the developer metrics are also created from these activities. In general, the metrics are related to the features of distributed code development processes such as issue, commit, and PR. We present some metrics that have been discussed in the literature in terms of these features and explain the challenges on which studies using these metrics have focused.

PR allows users to inform others about changes they have pushed to a branch in a repository on GitHub. PRs are a key feature when contributing code by different developers to a single project [11]. The proposed metrics related to this feature are used to solve different PR-related problems. PRs need to be reviewed (by a reviewer) in order to merge projects. If the result of a review is positive, the PR is integrated into the master branch. Finding the correct reviewer is thus an important parameter for ensuring rapid and fair PR revisions. Different metrics have been used in this context to address the problem of automatic PR reviewer assignment. The existing literature has proposed various metrics

to solve this problem, including PR acceptance rate within a project, active developers on a project [3], PR file location [12], pull requesters, social attributes [13], and textual features of the PR [14], among others. Closing a PR with an issue, PR age, and mentioning (@) a user in the PR comments have all been used to determine the priority of a PR [15]. Cosentino proposed three developer metrics (community composition, acceptance rates, and becoming a collaborator) to investigate project openness and stated that project owners could evaluate the attractiveness of their projects using these metrics [16].

Developer metrics are also used for software defect detection. In one study, defects were estimated using different metrics grouped by file and commit level. The number of files belonging to a commit, the most modified file of all files in a commit, the time between the first and last commit, and the experience of a given developer on a committed file were identified as important metrics [6].

Reliability metrics are used to quantitatively express the software product reliability [17]. Metrics used to measure reliability in OSS include number of contributors, number of commits, number of lines in commits, and certain metrics derived from them. Tiwari proposed two important metrics for reliability, namely contributors and number of commits per 1 K code lines [18].

Code ownership metrics are also important problem for OSS. One study used the modified (touched) number of files to rank developer ownership according to code contributions [19]. In another study, researchers used the number of changed lines in a file (churn) to address this problem [20]. Foucault confirmed the relationship between these code ownership metrics and software quality [21].

Recommender systems are an important research topic in software engineering [22,23]. In ordinary recommendation models, previously known user-item matrices are used. In other words, the rating given by a user for an item is known. In such cases, the essential research topic involves using different algorithms and models to estimate the rating that the user has already given [24]. However, this differs on software collaboration platforms like GitHub. Considering the developer as the user and the project (repository or repo) as the item, the rating that a developer gives to a project is unknown. In this context, the first problem that must be solved is how to create an accurate developer-project matrix. At this point, different developer metrics come into play.

In a study aiming to predict whether a user would join a project in the future, the metrics used included a developer's GitHub age (i.e., when their account was opened), the number of projects that they had joined, the programming languages of their commits, how many times a project was starred, the number of developers that joined a project, and the number of commits made to a project [25].

In another study that explored the factors that led a user to join a project, the metrics used included a developer's social connections, programming with a common language, and contributions to the same projects or files [26]. Liu et al. designed a neural network-based recommendation system that used metrics such as working at the same company, previous collaboration with the project owner, and different time-related features of a project [27].

Sun et al. relied on basic user activity to develop a project recommendation model using GitHub data. Specifically, when rating a project for a developer, they used "like-star-create" activities related to projects [5].

To sum up, some developer metrics that were noted as considerable in their papers are given in Table 1. In this table, we have also presented the fields of the metrics (the related feature), their definitions, and the target challenge, along with the reference studies. These metrics have been used to address various challenges in OSS development, among them project characterization, reviewer assignment for issues or PRs, and project recommendation are prominent ones. Generally, researchers have aimed to use these metrics to characterize developers by analyzing their activities in order to solve a problem [28]. Thus, developer metrics become crucial factors for solving these challenges. In this study, we offer a number developer metrics that can be used for a variety of challenges.

Table 1. Used metrics samples in the literature with GitHub data.

	Metric Name	Definition	Target Challenge	Ref
Issue	Maintenance Type	The type of issue that related with (feature, bug, etc.)	Finding issue resolution time factors	[29]
	A_D_issue_rep_assi	The total number of issues that assigned to specific developers	Understanding issue closure rates	[30]
	ContainsFix	Is the pull request whether solve an issue?	Pull request prioritization	[15]
	Owned_issues	The number of issues which is opened by the project's owner	Prediction of joining a repository	[25]
Commit	D_languages	The committed programming languages of a developer		[25]
	Contributors_SLOC	The number of contributors per 1K code lines	Exploring of reliability metrics	[18]
	TotalLoc	The number of lines in the last commit	Defect prediction with developer experience	[6]
	ExpLoc	The code line experience of a developer on a committed file		[6]
	ExpCom	The experience of a developer on a committed file		[6]
	NumofFile	The number of modified file in the last commit		[6]
	Past Experience	The number of files committed the past same-language.	Finding factors of joining a project	[26]
	Readme tf_idf	The tf-idf values of ReadMe and code files of project	Project recommendation	[5]
Pull Request	NumCommits	The number of commit in a PR	Automatic assignment of integrators to PR	[3]
	AcceptanceRate	The acceptance rates of PRs on a project		[3]
	TotalLines	The number of changed lines in a PR		[3]
	Age	Time between opening and closing a PR	Pull request prioritization	[15]
	FileLocation	The directory of files in a PR	Automatic code-reviewer assignment	[12]
	DecisionTime	PR decision time	Exploring the openness of a software project	[16]
	Files_changes	The number of changed files (some vs many)	Understanding the factors of assigning PR reviewers	[31]
	Comment Network	The network of pull request comments	Discovering factors of the PR process	[32]

3. Research Design

3.1. Dataset

One of the most serious challenges in developing a recommender system is sparsity [33], a problem that occurs when most users rate only a few items [34]. This issue is also present on GitHub, as it is not possible for developers to be aware of most of the millions of repositories on the platform. Most of the studies mentioned in the previous section used limited (less sparse) and ad hoc (unpublished) datasets (Table 2). In Table 2, we present the number of users, the number of projects, and the ratio of them (user/project or project/user) in the datasets of ours and others. Higher ratios indicates sparsity in the dataset. The greater a dataset's sparsity, the more difficult it is to make the correct recommendations. Thus, the results of the related studies are controversial in terms of real platform data (because of working on a smaller dataset). Therefore, in this study, we used a public dataset called GitDataSCP (<https://github.com/kadirseker00/GitDataSCP>) that is reflective of the sparsity problem inherent in the nature of GitHub [35].

Table 2. The dataset sparsity of related studies.

Paper ID	Number of User	Number of project	Ratio (~)
[5]	1700	22,000	0.07
[26]	1255	58,092	0.02
[27]	1070	1600	0.66
[25]	62,607	9447	0.15
[30]	62,607	9447	0.15
Our study	100	41,280	0.002

The dataset contained data related to 100 developers and 41,280 projects (repositories). The creators of the dataset indicated that they selected the most active users on the platform and extracted some related data from the activities of these GitHub users (Figure 1).

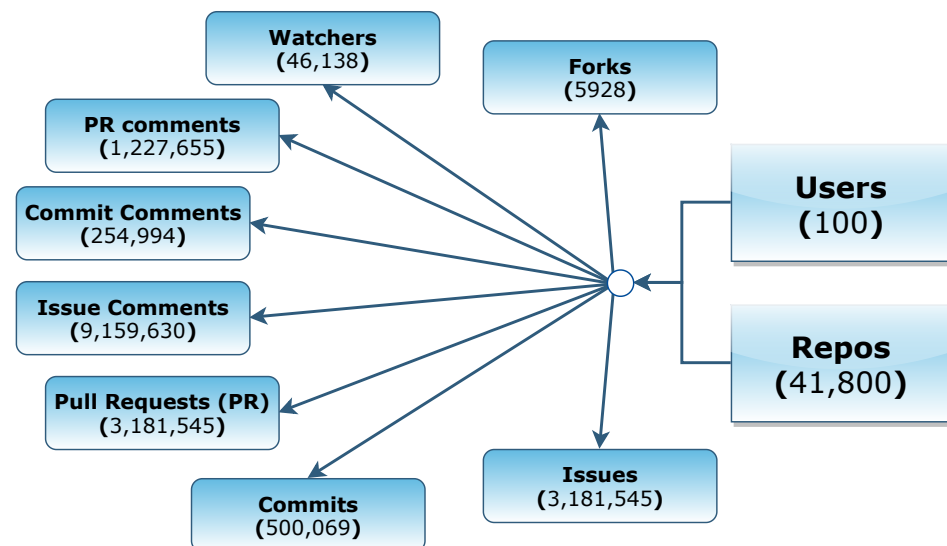


Figure 1. The dataset content.

The number of records in the dataset is also given in Figure 1 (below the name of collections). The *Repos* and all other collections include records that related to the *Users* collection. All details regarding the creation of the dataset are provided in the source study of the dataset [35].

3.2. Recommendation Model

We created a recommendation model based on item-item collaborative filtering. The collaborative filtering method usually involves gathering and analyzing information about a user's behavior, activities, or preferences and predicting what they will like based on their similarity to other users. Collaborative filtering is based on the assumption that different individuals who have had similar preferences in the past will make the same choices in the future. For example; if a user named John prefers A, B, and C products, and a user named Alice prefers B, C, and D products, it is likely that John will also prefer product D and Alice will prefer product A. Item-item logic can thus recommend similar items to a user who consumes any item.

Constructing a recommendation model for GitHub is different from classic (movie) principles. When recommending a movie to a user, the ratings of movies are known. However, this is not the case on GitHub, where developers do not rate projects. In this context, it is thus necessary to determine a metric that can be used as a rating. In addition, the model should recommend projects with which a developer is unfamiliar (just as a movie recommender should suggest movies that the user has not watched). We constructed a model taking into account these conditions.

A project recommender system for software collaboration development platforms includes stages as generating a project-developer rating matrix according to a certain metric, finding similarities between projects, generating recommendations, and evaluating results. First, we generated a project-developer rating matrix (Table 3) using specific metrics. As seen in Table 3, columns represent the users, rows represent the projects (repos). We selected a metric, then, input the metric's values (quantity, ratio, or binary) into the cells. For example, as shown in Table 3a, *User-1* opened seven issues in *Project-2*.

Table 3. A sample project-developer matrix belongs to the metric *issue_opened*.

(a) Actual values of the metric				
	User-1	User-2	...	User-100
Project-1	0	51	...	96
Project-2	7	0	...	0
Project-3	0	0	...	4
...
Project-n	0	34	...	0
(b) Normalized values of the metric				
	User-1	User-2	...	User-100
Project-1	0	4	...	10
Project-2	0.08	0	...	0
Project-3	0	0	...	0.5
...
Project-n	0	3	...	0

We normalized the values of the project-developer matrix using min-max normalization. The values of the metrics were scaled from 0 to 10. As in the movie-user model, we assumed that each developer gave a rating (0–10) to each project (Table 3b). First, we calculated similarity scores between projects using two methods (cosine and TF-IDF similarity). We then recommended the top-*n* projects to each developer. Finally, we evaluated the correct recommendations using two evaluation methods (community relation and language experience).

1. We obtained all comments that belonging to commits, issues, and PRs from all projects. Table 4 presents samples from issue comments.

Table 4. Issue comments at a glance.

User id	Issue id	Project id	Body
55859	138	1	I'd like to see this commit, or at least this feature, included. I'm trying to build something which uses the time information for a distributed team and I'd like to retain the timezone information . . .
145667	629	27	Here is a simpler example that I think shows the problem better. \r \n require fubinius/debugger . . .
...

2. We applied text preprocessing to these documents.
 - Convert all text into lower case.
 - Remove all digits.
 - Remove all punctuations.
 - Remove stopwords.
 - Remove extra whitespace.
3. We grouped all comments by projects and merged all comments into a single field (using the *comments* column of Table 5). We thus aimed to generate one comment document for each project.

Table 5. A sample project-developer matrix belongs to metricX

	Comments	Commit Comments	Issue Comments	Pull Request Comments
Project-1	like included exception . . .	exception . . .	like included . . .	NaN
Project-2
...	NaN
Project-n	...	NaN

4. The documents we generated were too large to be processed by an ordinary PC (The average word count for each project's issue comments was approximately 11,000). Therefore, we decided to select the most n frequent words for each comment. We applied Zipf's law to determine the cutoff point (n) [37]. As seen in Figure 3, we selected words with a rank greater than 100 and right of the second knee (function words). Thus, we generated documents for all projects that each included a maximum of 100 words.

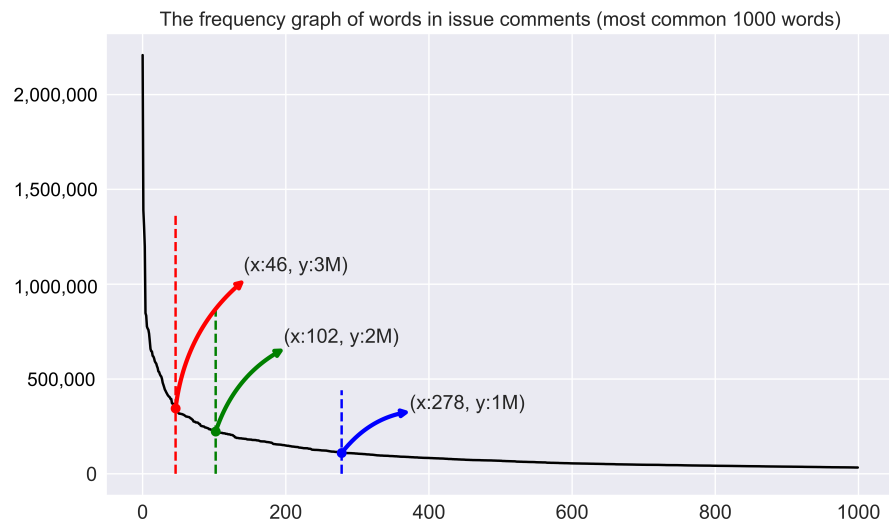


Figure 3. The zipf distributions of words in comments.

- We calculated the *TF-IDF* similarity scores for all projects with the documents generated in the previous step using the Equations (3) and (4).

$$tfidf_{(t,d,Project_i)} = tf(t, Project_i) * idf(t, Project_i) \tag{3}$$

$$similarity_{(P_i,P_j)}^{context_based} = tfidf(Project_i) * tfidf(Project_j) \tag{4}$$

3.3.3. Handling Unknown Projects

After we calculated similarities using above two methods, we generated ratings of unrated (unknown) projects for each developer. The similarity between unknown projects (We assumed that *unknown* projects were those to which developers had no relationship and had made no contributions.) and rated projects was used to calculate the rating of known projects [5]. We calculated the rating of an unknown project using the dot product of the similarity values between the projects that the user rated and the unknown project (Equation (5)). An example scenario involving this calculation is presented in Figure 4.

$$unknown_{rating} = \sum_{n=0}^i known^i_{rating} * similarity_{known^i,unknown} \tag{5}$$

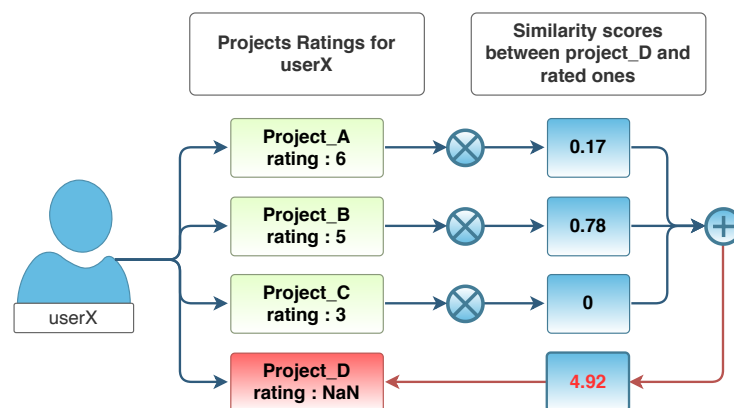


Figure 4. Calculating unrated project with the help of similarity rated projects.

3.4. Evaluation Methods

RQ1: Can we offer new evaluation methods for GitHub project recommendation problem?

We recommended the top- n highest-rated projects among the unknown projects to each developer. We then evaluated the recommendations using two methods we proposed. When recommending projects to developers, there should be a ground truth for evaluating the proposed projects. Unlike ordinary recommender systems, this is an unsupervised model. The evaluation criteria used in some studies related to this subject are set forth below.

- A project-user rating matrix was split randomly into test and training subsets. Accuracy or recall scores were then calculated from the intersection between the top- n scores of the test and train subsets [5]. However, another study argued that this method should not be used on platforms like GitHub where time is an important parameter, pointing out that problems would arise regarding predicting past activity with future data will occur when using k-fold cross-validation by randomly dividing the data [3].
- In another study, the accuracy of project recommendations was evaluated using the developer's past commits to the related project. A recommendation was assumed to be correct if the number of commits a certain developer made to the project exceeded a certain value. The average number of commits per project was set as the threshold value in the dataset [27].
- In a study predicting whether a developer would join a project in the future, the dataset was split into two different sets by time. In this way, the predicted result was verified with actual future data [25].
- In a survey-based study, the authors asked respondents which features could be used as a recommendation tool. Most of them stated that the languages in which developers already coded or with which they were familiar were important for recommendations [38].

In this study, we used two evaluation methods to analyze our proposed developer metrics.

3.4.1. Community Relation Approach

First, with the community relation approach, we used GitHub's *watching* and *forking* features as the ground truth. GitHub users can follow, or watch projects whose developments they want to monitor [39]. If a developer is watching a project, this indicates that he or she is interested in the project. Similarly, *forking* is used to contribute independently to the project of interest [40]. Developers usually make changes in their forked project (local branch) and can then send their contribution via PRs to base project (master branch). External developers mostly use the fork-pull mechanism to contribute to projects of interest. In this context, we believe that both of features are important for recommendations. While we used "*watching* or *forking*" as an evaluation criteria, we fine-tuned the criteria as detailed below.

The full name of a GitHub repository is created by concatenating the owner's name (login) with the repository name (e.g., davidteather/handracking). In analyzing our results, we noticed that the model recommended some project to developers that had only the correct owner login or repository name. In other words, the model suggested an incorrect project of the correct owner or the exact opposite. We evaluated these suggestions as a *half point* (0.5), as recommending only the correct owner to a developer will still allow the developer to learn other projects by that owner. Similarly, if the model recommends only a correct repository name with an incorrect owner, this indicates that it has suggested the forked version of a correct repository. Thus, the related developer can discover with the master (base) repository.

An example scenario demonstrating this situation is given in Table 6. The projects recommended for Alice are listed in the first column Table 6a. Two of them are among the repos that Alice watches, with the two correct matches, the initial score is 2. There are two repos by a developer named *fengmk2* among Alice’s watched projects (*fengmk2/parameter* and *fengmk2/cnpmjs.org*) (Table 6b). The model suggested the project *fengmk2/emoji* that belongs to a developer who Alice is familiar with him. Similarly, Alice watches a forked project of *visionmedia/co*. Thus, two *half*-score are added to the initial score and 3: (2 + 0.5 + 0.5) is the final score.

Table 6. A sample scenario for recommending with correct and half matches.

(a) Recommendations for Alice		
Top-5 recommendations	Correct matches	Half matches
visionmedia/ <u>co</u>		co
fengmk2/emoji		fengmk2
iojs/io.js	iojs/io.js	
julgruber/co-read	julgruber/co-read	
koajs/compose/		
(b) Alice’s watched repos		
Score ^a	Repo full names	
⊕	adamwiggins/ co	
⊕	fengmk2/parameter	
⊕	iojs/io.js	
⊕	julgruber/co-read	
⊖	fengmk2/cnpmjs.org	
...	...	

^a ⊕: correct, ⊕: half, ⊖:wrong.

We used the Equation (6) to calculate hit score (In other words, the scores of correct recommendations). Our analysis showed that some developers had only a few watched projects. Thus, the case of a developer interested in (watching or forking) fewer than *n* projects was considered in the updated score Equation (6).

$$hit_{score}^{community} = \begin{cases} 100 * \frac{hit_{fullname} + (hit_{partial} * 0.5)}{n}, & \text{if } num_{watch_or_fork} \geq n \\ 100 * \frac{hit_{fullname} + (hit_{partial} * 0.5)}{num_{watch_or_fork}}, & \text{otherwise} \end{cases} \quad (6)$$

To sum up, in this evaluation approach, if the recommended project is among the developer’s watched or forked projects, the project is considered a hit.

3.4.2. Language Experience Approach

With our second approach, we wanted to benefit from the developer’s coding language experience. Thus, we processed the languages of projects and used the knowledge obtained as an evaluation criterion. The projects in our dataset included 94 unique coding languages. The most used 20 languages are shown in Figure 5. Due to the diversity of languages in the dataset, we believe that the language feature can be used as another evaluation criterion.

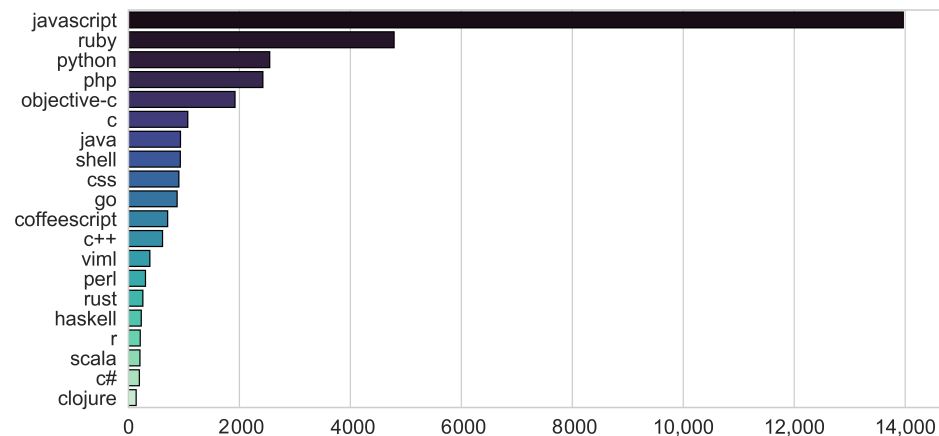


Figure 5. The most common used languages in the dataset.

We extracted all programming languages for projects that developers owned or watched or to which they made commits (Table 7). We aimed to discover the languages in which a developer had any activity. We then sorted them by frequency and identified the three most used languages (“expert languages” column in Table 7).

Table 7. The coding languages that experienced in past for each developer.

User id	All Languages	Expert Languages
21	[c, ruby, ruby, ruby, go, ruby, ruby, ...]	[ruby, javascript, go]
13760	[objective-c, objective-c, c, haskell, ...]	[objective-c, c, ruby]
3346407	[css, python, python, lua, c++, ...]	[python, javascript, css]
...

Similarly, in this evaluation approach, if the recommended project’s language was among a developer’s top languages, the project was considered a hit. This evaluation’s scores were considerably higher than the first’s. However, our aim was to validate the significant metrics with another evaluation criterion.

$$hit_{score}^{experience} = 100 * \frac{hit_{language}}{n} \tag{7}$$

In this way, we created a project recommendation model has been created for software collaboration platforms. The algorithm of the recommendation model is presented in Figure 6, starting with selecting a feature as a metric and ending with calculating hit scores.

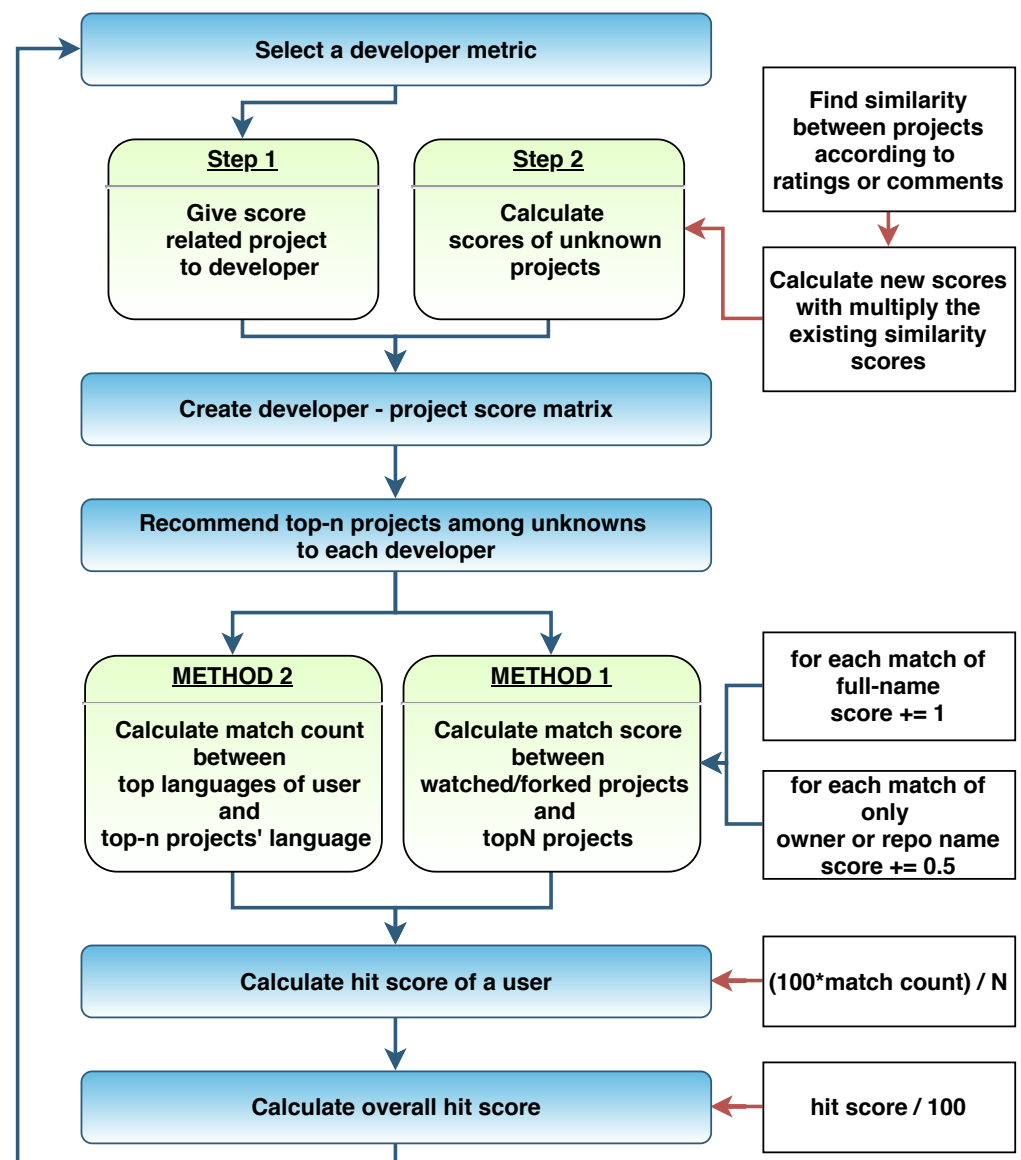


Figure 6. The flowchart of project recommendation system.

4. Empirical Results

We generated 40 different developer metrics that provide information about a developer's past activity on a project. All metrics used were scaled from 0 to 10 using the min-max normalization technique. The project-developer relationship was thus rated in the range of 0 to 10 (as with a viewer's rating of a movie). We then applied all of these metrics to the project recommendation model and evaluated the results with the top 1, 3, 5, 10, and 20 recommendations hit scores.

4.1. Generating Developer Metrics

We created developer metrics using several methods. To extract metrics for a developer in a project, we used the number of activities, the ratio between some number of activities, and the status of whether an activity exists or not. First, using activities individually, we created metrics called single metrics. We then combined the single metrics according to common features to obtain the fusion metrics. Lastly, we created binary fusion metrics indicating whether a particular activity existed in a project.

4.1.1. Single Metrics

RQ2: Can we use the activities of GitHub users as a developer metric individually?

Developer activity on projects was handled as a metric. Activity includes all kinds of comments, code contributions, revisions, and so on. In this section, all metrics were treated individually in order to evaluate the significance of each. These metrics refer to the number of activities per project for a given developer (Table 8).

Table 8. Single developer metrics definitions.

	Metric	Definition
1	issue_opened	Number of issue opened
2	issue_commented	Number of comment to issue
3	issue_closed	Number of issue closed
4	issue_hasPR	Number of issue that has a PR
5	issue_assigned	Number of issue assigned
6	commit_commented	Number of comment to commit
7	commit_authored	Number of authorship in commit
8	commit_committed	Number of commit
9	pr_opened	Number of pull requests
10	pr_merged	Number of PR merged
11	pr_assigned	Number of PR assigned
12	pr_commented	Number of comment to PR

In addition to these metrics, we calculated the optimized metrics from the values of single developer metrics. We named these metrics with the prefix 'O_'. For example, name of optimized *pr_closed* is *O_pr_closed*. We aimed to show the contribution ratio of each user for each project. For example, John closed 10 PRs in a projectA, and 10 in projectB. The total number of closed PRs is 100 in projectA and 1000 in projectB. Therefore, John contributed much more to projectA (10% contribution) than to projectB (1% contribution), as his closing PR ratio in projectA is higher. We calculated the rating with Equation (8).

$$O_{activity}_x = \frac{\#_of_activity_x^{user}}{\#_of_total_activity_x^{project}} \quad (8)$$

For comparison purposes, we added another metric proposed by Sun et. al. They scored developers and projects using *like*, *star*, *create* activities and used textual data extracted from projects' README and source code files to find project similarities [5]. Their dataset included approximately 22,000 repositories and 1700 developers. In our dataset, the ratio of number of developers to number of projects was approximately 1:400, in Sun et al.'s study it was 1:14. We also planned to use this less sparsed dataset to make a fair comparison but could not because the dataset was unshared. As we were unable to communicate with the authors, we applied a very similar rating to our dataset.

4.1.2. Fusion Metrics

RQ3: Can new metrics be obtained by combining similar properties or activities?

In our results, we observed that some metric groups came to the forefront, especially issue related metrics. New metrics can be proposed by grouping comments, code contributions, or other common feature metrics. Fusion metrics were created from combinations of single metrics.

1. *Sun's metric*: is mentioned previous section. It is metrics from similar study [5].
2. *code_contributions*: is created from the sum of all code contribution-related metrics.

$$code_contributions = pr_opened + issue_opened + issue_hasPR + pr_merged + commit_committed \quad (9)$$

3. *comments*: is created from the sum of all comment-related metrics.

$$comments = issue_commented + commit_commented + pr_commented \quad (10)$$

4. *issue_related*: is created from the sum of all issue-related metrics.

$$issue_related = issue_opened + issue_hasPR + issue_commented + issue_assigned \quad (11)$$

5. *pr_related* is created from the sum of all PR-related metrics.

$$pr_related = pr_opened + pr_merged + pr_closed + pr_assigned \quad (12)$$

6. *commit_related*: is created from the sum of all commit-related metrics.

$$commit_related = commit_commented + commit_authored + commit_committed \quad (13)$$

7. *commit2comment* is created from the (*commit_committed* divided by *commit_commented*)

$$commit2comment = \frac{commit_committed}{commit_commented} \quad (14)$$

8. *issue2comment* is created from the (*issue_opened* divided by *issue_commented*)

$$issue2comment = \frac{issue_opened}{issue_commented} \quad (15)$$

9. *pr2comment* is created from the (*pr_opened* divided by *pr_commented*)

$$pr2comment = \frac{pr_opened}{pr_commented} \quad (16)$$

10. *code2comment* is created from the ratio of two fusion metrics (*contribution* divided by *comment*)

$$code2comment = \frac{contribution}{comment} \quad (17)$$

4.1.3. Binary Fusion Metrics

RQ4: Is a user's amount of activity important in the context of developer metrics?

The above metrics offer information about how many activities were made. For instance, if John opened 18 issues in projectX, the John-projectX rating is 18. As an alternative, a set of metrics was created that simply showed whether a given activity existed. For instance, even so, if John opened an issue in projectX, the John-projectX rating is 1; if John did not open an issue in projectY, the John-projectY rating is 0. We created the *binary metrics* using the Equation (18) from the fusion metrics.

$$BinaryMetrics = \begin{cases} 1, & \text{if } SingleMetric > 0 \\ 0, & \text{otherwise} \end{cases} \quad (18)$$

Because of the binary metrics consisted only of 0 s and 1 s, we did not used them directly. Instead, we created binary fusion metrics. We named these metrics with the prefix 'binary_'; for example, the name of the binary fusion metric for *comments* is *binary_comments*. from the binary metrics using the same equations while creating fusion metrics from the single metrics.

4.2. Project Recommendation Results

In this section, we showed the only top 1, 5, and 10 hit scores of the most significant metrics. Most of the ratio-based fusion metrics and optimized single metrics were very weak in all cases. Therefore, we removed them from the score tables below. The results of all metrics according to all n values are provided in the Appendix A.

We used two similarity metrics to calculate project’s similarity scores. To evaluate the accuracy of developer metrics, we used from two approaches. Thus, we give four results for the recommendation system with the combinations of similarity methods and evaluation approaches (Figure 7).

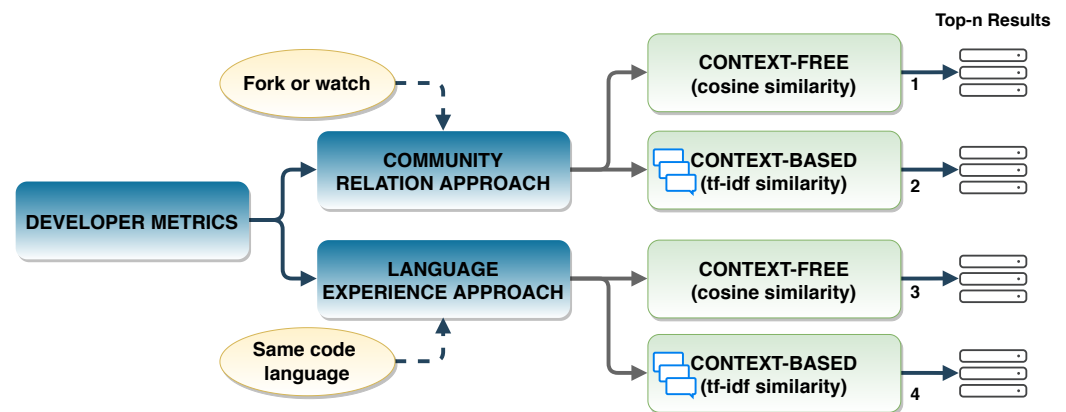


Figure 7. The hit scores calculations with 4 different parameters combinations.

After we generated these single, fusion, and binary fusion metrics, we applied all developer metrics to the model. We presented the hit scores percentiles that were obtained according to the two evaluation approaches (detailed in the Section 3.4) in Tables 9 and 10.

In these tables, the columns represent the top- n hit scores in percentiles, and the green (1st), blue (2nd), and red (3rd) cells show the leading metrics in each top- n scores. In addition, we styled the most successful five metrics according to overall scores in **bold**. We used the mean reciprocal rank (MRR) (It is commonly used in question-answering systems. Here, we used number of models instead of number of queries.) evaluation method (Equation (19)) to calculate this overall score where n represents the number of models created ($n:6$).

$$score_{MRR} = \frac{1}{n} \sum_{i=0}^n \frac{1}{rank_i} \tag{19}$$

For example, as shown in Table 9, the MRR value of *comments* is calculated as in Equation (20). The metric ranks for all metrics were used in each of the six models.

$$score_{MRR}^{comments} = \frac{1}{6} * \left(\frac{1}{2} + \frac{1}{1} + \frac{1}{2} + \frac{1}{15} + \frac{1}{16} + \frac{1}{18} \right) = 0.36 \tag{20}$$

In addition to Tables 9 and 10, we presented all top- n hit scores charts of the most successful five metrics (+1 Sun_metric) for each methods in Figures 8 and 9.

Table 9 shows the results of hit scores using the context-free and context-based similarity methods as evaluated using the community relation approach. When the results were analyzed, the *pr_opened* metric was the most successful according to MRR scores. As PRs indicate projects to which a developer contributed directly, we believe that modeling based on PR creation in a given project increases the success of the project recommendation system. In addition, the fusion metrics *comment*, *binary_pr_related*, *binary_issue_related*, and *binary_comment* also attracted our attention, as these metrics all related to commenting activity. The results therefore indicate the importance of discussion on the collaboration platforms.

Table 9. Developer metrics scores according to evaluate with the community relation approach.

ID	Developer Metrics (community approach)	Context-free similarity			Context-based similarity			MRR
		top1	top5	top10	top1	top5	top10	
1	commit_authored	14.00	12.70	11.55	28.50	23.10	18.45	0.10
2	commit_commented	22.00	19.20	16.95	26.00	20.50	18.45	0.08
3	commit_committed	15.50	12.90	11.55	29.00	22.60	18.50	0.11
4	issue_assigned	7.00	8.30	7.50	26.50	17.40	14.25	0.06
5	issue_closed	11.50	8.10	6.25	20.00	14.80	13.15	0.05
6	issue_hasPR	26.00	17.90	16.20	34.00	24.20	21.70	0.25
7	issue_commented	30.00	25.50	23.80	22.00	18.20	16.95	0.20
8	issue_opened	23.00	20.70	18.70	26.00	22.10	18.65	0.10
9	pr_assigned	4.00	4.20	4.30	12.50	9.30	8.50	0.04
10	pr_commented	20.00	19.60	17.85	20.00	20.10	18.60	0.07
11	pr_merged	16.00	12.80	11.80	28.00	20.70	18.90	0.09
12	pr_opened	26.10	19.20	16.00	35.50	24.70	22.30	0.48
13	comments	31.00	26.50	24.15	22.00	19.30	16.95	0.36
14	code_contributions	23.00	22.20	21.00	24.00	20.50	18.25	0.09
15	commit_related	23.00	18.40	17.75	26.00	19.40	19.00	0.08
16	pr_related	25.00	23.60	22.05	22.00	23.90	20.10	0.15
17	issue_related	25.00	24.40	23.35	18.50	17.00	15.20	0.11
18	binary_commit_related	24.00	19.30	19.10	31.00	23.00	20.20	0.17
19	binary_pr_related	23.50	22.50	22.35	28.50	27.50	21.90	0.34
20	binary_issue_related	25.50	25.90	24.90	15.00	17.70	16.40	0.30
21	binary_code_contributions	24.00	24.40	23.55	22.00	20.70	18.05	0.12
22	binary_comments	32.00	24.80	23.35	17.50	18.80	17.60	0.26
23	O_issue_commented	22.00	21.20	19.25	15.00	13.00	12.25	0.06
24	code_2_comment	22.50	19.80	17.85	27.50	22.20	19.40	0.11
25	Sun_metric	8.00	7.60	7.75	14.50	11.50	9.90	0.04

Degrees: 1st 2nd 3th

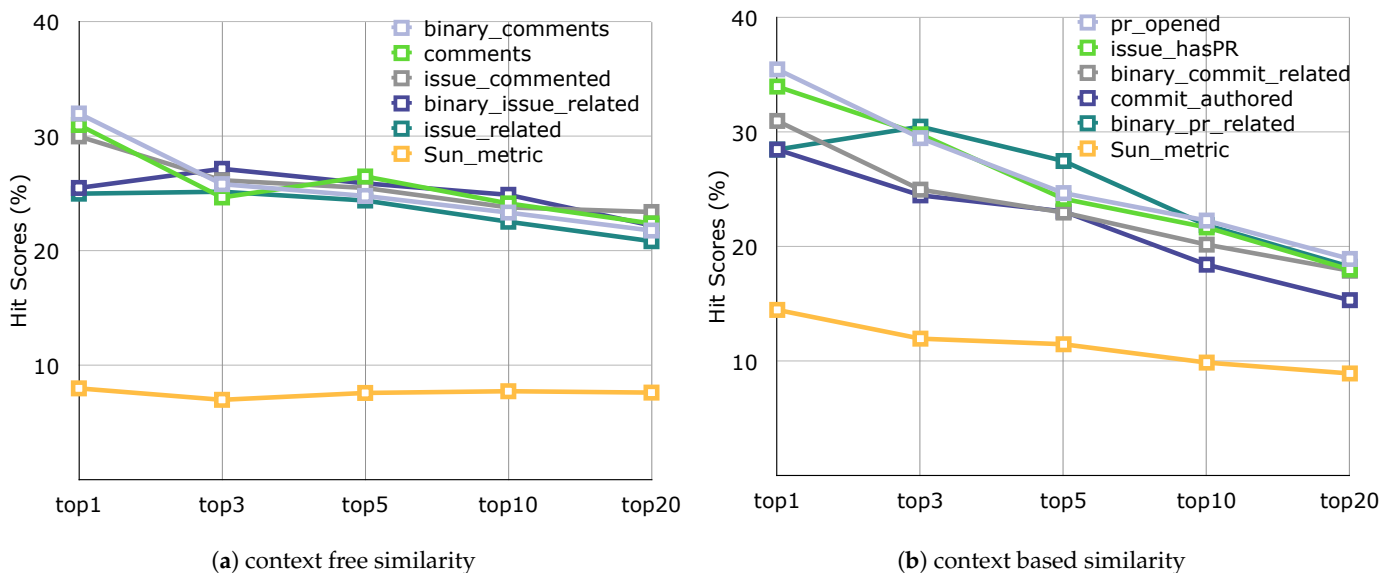


Figure 8. The most successful 5 developer metrics all hit scores (community).

Table 10 shows the results of hit scores according to the context-free and context-based similarity methods as evaluated using the language experience approach. This approach elicited higher hit scores than the first approach as we expected. (In the community relation approach, the model recommends the top-*n* projects of the approximately 40,000 projects. On the contrary, the language experience approach makes only recommendation from 94 distinct language projects). However, it is important that evaluate the success rate of the two approaches independently. The binary fusion metrics clearly stand out and most fusion metrics gave better hit scores than single metrics. In this context, because a fusion

metric represent many features about a developer, we believe that projects developed in the languages in which the developer specializes are better explored. In addition, in this approach, the *binary_issue_related* metric leads among the fusion metrics. The issue that is the primary function of project management operations is the most crucial feature in terms of developer-project correlation. Lastly, the results of *Sun_metric* drew attention here, unlike with the previous approach.

Table 10. Developer metrics scores according to evaluate with the language experience approach.

ID	Developer Metrics (experience approach)	Context-free similarity			Context-based similarity			MRR
		top1	top5	top10	top1	top5	top10	
1	commit_authored	23.00	35.60	36.30	65.00	58.80	57.00	0.05
2	commit_commented	74.00	57.60	51.90	72.00	66.60	63.90	0.10
3	commit_committed	23.00	35.60	35.30	65.00	57.60	55.70	0.04
4	issue_assigned	64.00	52.20	53.80	59.00	57.80	53.30	0.05
5	issue_closed	68.00	59.60	55.40	50.00	49.40	46.40	0.05
6	issue_hasPR	72.00	62.60	61.10	73.00	70.80	69.00	0.09
7	issue_commented	66.00	66.80	65.60	75.10	70.60	70.20	0.12
8	issue_opened	68.00	67.60	66.40	75.00	71.80	70.70	0.14
9	pr_assigned	44.00	26.60	20.50	36.00	30.00	29.80	0.04
10	pr_commented	48.00	47.80	48.00	60.00	61.40	60.10	0.05
11	pr_merged	74.00	50.40	45.00	65.00	62.20	60.30	0.08
12	pr_opened	70.00	61.80	61.10	70.00	69.20	68.30	0.08
13	comments	67.00	63.40	63.70	76.00	72.00	70.60	0.18
14	code_contributions	75.00	67.00	64.90	70.00	70.40	70.80	0.15
15	commit_related	69.00	61.80	60.10	72.00	67.60	64.50	0.07
16	pr_related	63.00	62.80	64.00	69.00	71.20	67.60	0.08
17	issue_related	74.00	71.80	67.30	76.00	71.40	69.60	0.20
18	binary_commit_related	72.00	64.60	60.40	70.00	69.20	65.90	0.08
19	binary_pr_related	70.00	67.00	66.50	78.00	77.40	74.40	0.45
20	binary_issue_related	86.00	77.40	74.40	75.00	72.60	75.00	0.65
21	binary_code_contributions	73.00	72.60	70.70	76.00	73.40	74.30	0.30
22	binary_comments	77.00	68.60	70.10	75.00	74.40	75.20	0.43
23	code_2_comment	52.00	61.00	61.20	65.00	70.20	68.90	0.07
24	O_issue_closedPR	73.00	55.80	51.20	60.00	56.60	57.20	0.06
25	Sun_metric	73.00	74.00	70.70	68.00	71.40	72.00	0.22

Degrees:

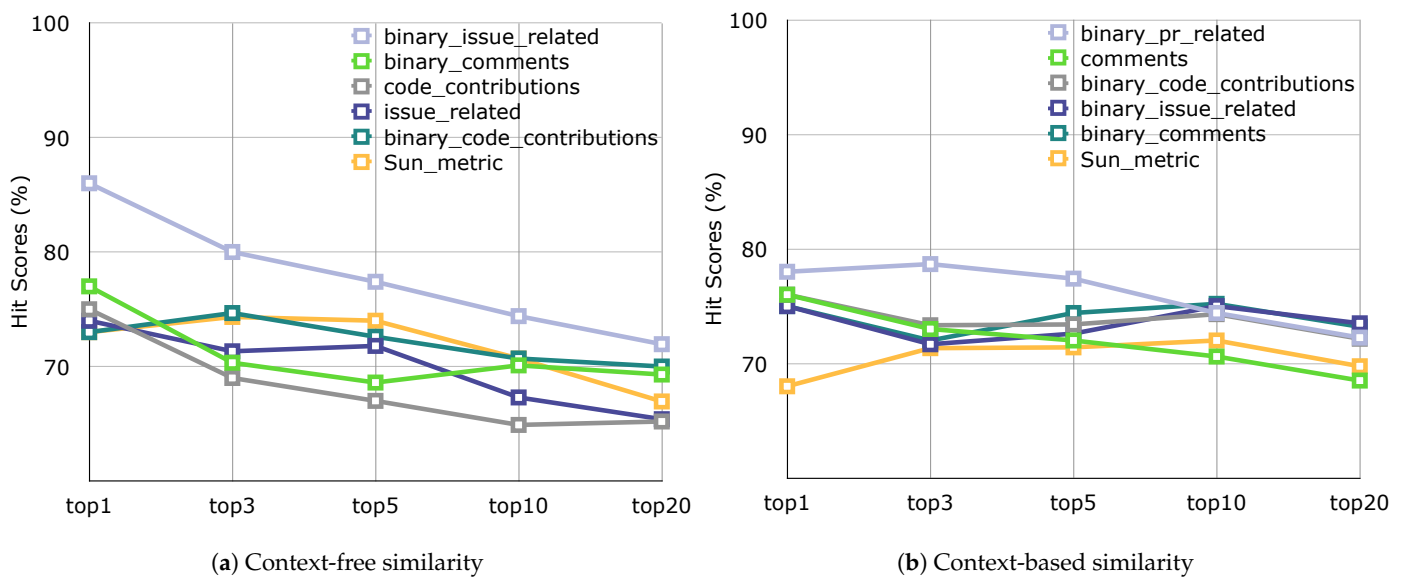


Figure 9. The most successful five developer metrics all hit scores (experience).

Analyzing both Tables 9 and 10 together, we saw that binary metrics were quite successful. It is also a noteworthy that comment-based metrics achieved higher scores than code activity-based metrics.

4.3. Threats to Validity

The scope of this study was limited to active developers on GitHub. The first challenge is whether these metrics will work well for inactive developers, a case resembling the cold start problem in classic recommender systems. It is even more difficult to make recommendations for inactive developers due to the comparative lack of information about them. Our proposed metrics must therefore be analyzed in other datasets that include such developers.

The project recommendation problem is important for collaboration platforms. GitHub has recently started to offer project recommendations on the “Explore” page based on certain user activities. Apart from the metrics we proposed, metrics applied to other challenges can successfully be used for the recommendation problem. We encourage researchers to work on this problem using different metrics.

Another problem involves studying private datasets for software engineering challenges. Making comparisons to studies that use different datasets can be challenging. In this sense, our results are limited to our own dataset (which is public). Finally, unlike classic recommender systems, there are no labeled data (ground truth) for our problem. For this reason, we consider it important to create a labeled dataset that can be used to work on the project recommendation systems for platforms such as GitHub.

5. Conclusions

We extracted different types of metrics using the number of activities, the ratio of some metrics, and only the case of whether activity exists. To evaluate our metrics, we developed a top- n project recommender system based on collaborative filtering using item similarity logic which finds items similar to those with which a user has already interacted (e.g., liking, disliking, or rating). In our study, an interaction with an item refers to a contribution to a project. We used two different methods to calculate the similarity between projects. The context-based similarity method had a positive impact on the hit scores. We then evaluated the accuracy of metrics with two particular approaches.

In a movie recommender system, the ground truth is users’ actual ratings. However, there is no common evaluation baseline for GitHub project recommendation systems. Accordingly, in this paper, we proposed two approaches—community relation and language experience—as ground truth. The community relation approach checks whether a developer has watched (or forked) a given project, while the language experience approach uses as a baseline whether the language of the project is one in which the developer has previous experience.

First, we extracted developer metrics for individual activities. Among these single metrics, the most prominent were *pr_opened*, *issue_hasPR*, and *issue_commented*. In this context, we believe that these metrics are adequate even when used individually to obtain knowledge about developers. The crucial single metrics have common traits (such as the issue feature or commenting activity). Next, we created some fusion metrics by combining single metrics. Of these fusion metrics, the *comments* metric produced significant results. Lastly, as we were curious about whether the amount of activity was important in the context of developer metrics, we created the binary fusion metrics based on the case of activities existence. Taking all results together, the peak scores were gained from these metrics. In particular, the *binary_issue_related* and *binary_comments* were the most attention-grabbing metrics.

As PRs indicate projects to which a developer has contributed directly, we believe that modeling based on PR creation in a given project increases the success of the project recommendation system. The issue that is the primary function of project management operations is the most crucial feature in developer–project correlation.

Our results indicate that quantity is not a crucial parameter for some metrics. For example, the issue- and PR-related metrics are quantity-free metrics. Effectively, this means that, when using these metrics, it is sufficient to know whether the feature in question is present. Even if a developer contributes to only one issue on a project, the relation between the developer and the project is tight. In this regard, it is revealing that this issue was a significant feature for collaboration platforms.

In conclusion, we have proposed remarkable and improvable developer metrics based on user activities in GitHub. In particular, we found that commenting on any feature was as important as code contributions. Issue-related activities were also highly important in developer metrics.

We took into consideration the challenge of the sparsity inherent in the nature of GitHub. Despite the sparsity problem, our hit scores were notable compared with a similar study, with most of our metrics more successful than their metric.

Finding similarities between documents is very difficult. In future research, we plan to use word embedding (e.g., word2vec, GloVe, etc.) methods instead of TF-IDF. We plan to apply the proposed metrics to different datasets for validation purposes. We are curious about why some of the new developer metrics we presented became prominent. In light of this study, we are planning another study involving a survey of junior and senior developers whom we can contact to understand the ground truth of our metrics' success (especially metrics related to commenting activities). In addition, we plan to apply these metrics to solve various problems. For instance, many developers, in addition to owners and collaborators, can make contributions to projects thanks to the open-source nature of GitHub. On some projects, external developers even contribute more than the core team. These metrics can reveal developers' contribution rankings on a particular project.

Author Contributions: Conceptualization, A.Ş.; Data curation, H.A.; Formal analysis, A.Ş. and B.D.; Funding acquisition, A.Ş.; Methodology, A.Ş., B.D. and H.A.; Project administration, B.D.; Resources, A.Ş. and H.A.; Supervision, H.A.; Validation, A.Ş. and H.A.; Visualization, A.Ş.; Writing—original draft, A.Ş. and B.D.; Writing—review & editing, B.D. and H.A. <http://img.mdpi.org/data/contributor-role-instruction.pdf> (CRediT taxonomy) for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Table A1. All developer metrics all scores to evaluate with the community approach (context-free).

ID	Developer Metrics (community approach)	Context-free similarity					
		top1	top3	top5	top10	top20	mean
1	binary_code_contributions	24.00	25.33	24.40	23.55	22.08	23.87
2	binary_comments	32.00	25.83	24.80	23.35	21.78	25.55
3	binary_commit_related	24.00	21.83	19.30	19.10	17.30	20.31
4	binary_issue_related	25.50	27.17	25.90	24.90	22.22	25.14
5	binary_pr_related	23.50	21.83	22.50	22.35	21.08	22.25
6	code_2_comment	22.50	21.00	19.80	17.85	15.82	19.39
7	code_contributions	23.00	22.33	22.20	21.00	19.23	21.55
8	comment_2_code	24.00	18.17	17.20	16.35	14.70	18.08
9	comments	31.00	24.67	26.50	24.15	22.40	25.74
10	commit_2_comment	14.00	10.33	10.30	9.75	7.65	10.41
11	commit_authored	14.00	12.17	12.70	11.55	10.45	12.17
12	commit_commented	22.00	20.17	19.20	16.95	14.65	18.59
13	commit_committed	15.50	12.33	12.90	11.55	10.20	12.50
14	commit_related	23.00	20.17	18.40	17.75	16.18	19.10
15	issue_2_comment	23.50	20.17	17.50	16.50	14.58	18.45
16	issue_assigned	7.00	8.83	8.30	7.50	5.95	7.52
17	issue_closed	11.50	9.50	8.10	6.25	5.98	8.27
18	issue_hasPR	26.00	20.83	17.90	16.20	14.90	19.17
19	issue_commented	30.00	26.17	25.50	23.80	23.40	25.77
20	issue_opened	23.00	21.67	20.70	18.70	16.77	20.17
21	issue_related	25.00	25.17	24.40	22.55	20.85	23.59
22	O_commit_authored	14.50	11.67	12.60	11.45	10.20	12.08
23	O_commit_commented	19.50	18.33	18.20	17.25	15.20	17.70
24	O_commit_committed	15.00	11.67	12.80	11.15	9.90	12.10
25	O_issue_assigned	7.00	8.83	8.30	7.50	5.95	7.52
26	O_issue_closed	11.50	9.50	8.10	6.25	5.98	8.27
27	O_issue_closedPR	16.00	16.00	14.40	14.00	12.88	14.66
28	O_issue_commented	22.00	23.17	21.20	19.25	17.62	20.65
29	O_issue_opened	15.00	15.17	15.80	15.40	15.48	15.37
30	O_pr_assigned	4.00	5.33	4.20	4.30	3.95	4.36
31	O_pr_commented	21.50	19.00	20.10	18.15	17.10	19.17
32	O_pr_merged	15.50	12.33	12.50	10.90	10.65	12.38
33	O_pr_opened	20.00	16.50	15.30	13.60	12.10	15.50
34	pr_2_comment	14.50	14.83	15.00	14.05	11.58	13.99
35	pr_assigned	4.00	5.33	4.20	4.30	3.95	4.36
36	pr_commented	20.00	18.00	19.60	17.85	15.50	18.19
37	pr_merged	16.00	13.00	12.80	11.80	11.18	12.96
38	pr_opened	26.00	19.83	19.20	16.00	14.20	19.05
39	pr_related	25.00	24.17	23.60	22.05	20.80	23.12
40	Sun_metric	8.00	7.00	7.60	7.75	7.63	7.60

Table A2. All developer metrics all scores to evaluate with the community approach (context-based).

ID	Developer Metrics (community approach)	Context-based similarity					
		top1	top3	top5	top10	top20	mean
1	binary_code_contributions	22.00	23.00	20.70	18.05	16.50	20.05
2	binary_comments	17.50	18.17	18.80	17.60	15.40	17.49
3	binary_commit_related	31.00	25.00	23.00	20.20	17.92	23.42
4	binary_issue_related	15.00	17.00	17.70	16.40	14.98	16.22
5	binary_pr_related	28.50	30.50	27.50	21.90	18.27	25.33
6	code_2_comment	27.50	22.17	22.20	19.40	17.12	21.68
7	code_contributions	24.00	22.17	20.50	18.25	15.22	20.03
8	comment_2_code	25.00	22.83	23.20	20.25	18.10	21.88
9	comments	22.00	20.00	19.30	16.95	14.75	18.60
10	commit_2_comment	25.00	20.83	18.30	14.35	12.18	18.13
11	commit_authored	28.50	24.50	23.10	18.45	15.35	21.98
12	commit_commented	26.00	20.50	20.50	18.45	15.60	20.21
13	commit_committed	29.00	24.50	22.60	18.50	15.28	21.98
14	commit_related	26.00	20.17	19.40	19.00	16.12	20.14
15	issue_2_comment	24.50	18.67	19.80	17.35	16.68	19.40
16	issue_assigned	26.50	19.50	17.40	14.25	11.85	17.90
17	issue_closed	20.00	17.67	14.80	13.15	11.55	15.43
18	issue_hasPR	34.00	29.83	24.20	21.70	18.02	25.55
19	issue_commented	22.00	18.83	18.20	16.95	15.08	18.21
20	issue_opened	26.00	22.33	22.10	18.65	16.05	21.03
21	issue_related	18.50	16.50	17.00	15.20	13.68	16.18
22	O_commit_authored	28.00	22.17	20.10	17.70	15.12	20.62
23	O_commit_commented	21.50	19.33	17.80	16.65	15.82	18.22
24	O_commit_committed	25.50	24.33	20.50	18.10	15.18	20.72
25	O_issue_assigned	17.00	17.50	14.40	12.70	10.32	14.38
26	O_issue_closed	13.00	10.83	10.20	10.40	9.02	10.69
27	O_issue_closedPR	21.00	18.33	16.60	14.50	12.82	16.65
28	O_issue_commented	15.00	13.67	13.00	12.25	11.10	13.00
29	O_issue_opened	16.50	15.83	15.60	13.00	12.65	14.72
30	O_pr_assigned	14.00	10.00	9.10	9.60	7.78	10.10
31	O_pr_commented	24.50	24.33	21.50	19.70	16.68	21.34
32	O_pr_merged	19.50	16.67	14.70	13.45	13.35	15.53
33	O_pr_opened	17.50	18.17	17.00	15.25	13.40	16.26
34	pr_2_comment	23.00	21.00	18.90	16.20	15.05	18.83
35	pr_assigned	12.50	10.67	9.30	8.50	7.55	9.70
36	pr_commented	20.00	21.83	20.10	18.60	16.15	19.34
37	pr_merged	28.00	21.33	20.70	18.90	17.05	21.20
38	pr_opened	35.50	29.50	24.70	22.30	18.95	26.19
39	pr_related	22.00	26.50	23.90	20.10	17.02	21.90
40	Sun_metric	14.50	12.00	11.50	9.90	8.95	11.37

Table A3. All developer metrics all scores to evaluate with the experience approach (context-free).

ID	Developer Metrics (experience approach)	Context-free similarity					mean
		top1	top3	top5	top10	top20	
1	binary_code_contributions	73.00	74.67	72.60	70.70	70.00	72.19
2	binary_comments	77.00	70.33	68.60	70.10	69.30	71.07
3	binary_commit_related	72.00	65.33	64.60	60.40	57.20	63.91
4	binary_issue_related	86.00	80.00	77.40	74.40	71.95	77.95
5	binary_pr_related	70.00	69.00	67.00	66.50	64.85	67.47
6	code_2_comment	52.00	62.00	61.00	61.20	60.90	59.42
7	code_contributions	75.00	69.00	67.00	64.90	65.20	68.22
8	comment_2_code	59.00	61.67	61.40	62.30	61.15	61.10
9	comments	67.00	62.67	63.40	63.70	63.30	64.01
10	commit_2_comment	13.00	28.67	32.00	28.50	39.15	28.26
11	commit_authored	23.00	31.00	35.60	36.30	33.95	31.97
12	commit_commented	74.00	62.67	57.60	51.90	52.95	59.82
13	commit_committed	23.00	32.67	35.60	35.30	33.55	32.02
14	commit_related	69.00	63.67	61.80	60.10	57.45	62.40
15	issue_2_comment	65.00	66.67	64.60	64.50	63.05	64.76
16	issue_assigned	64.00	54.67	52.20	53.80	48.20	54.57
17	issue_closed	68.00	54.33	59.60	55.40	49.05	57.28
18	issue_hasPR	72.00	64.67	62.60	61.10	58.80	63.83
19	issue_commented	66.00	68.00	66.80	65.60	65.80	66.44
20	issue_opened	68.00	67.67	67.60	66.40	64.15	66.76
21	issue_related	74.00	71.33	71.80	67.30	65.40	69.97
22	O_commit_authored	23.00	31.67	36.00	37.10	34.05	32.36
23	O_commit_commented	73.00	60.33	55.80	51.20	53.15	58.70
24	O_commit_committed	22.00	32.00	35.20	34.80	33.05	31.41
25	O_issue_assigned	64.00	54.67	52.20	53.80	48.20	54.57
26	O_issue_closed	68.00	54.33	59.60	55.40	49.05	57.28
27	O_issue_closedPR	63.00	65.33	63.20	61.30	60.35	62.64
28	O_issue_commented	68.00	64.00	62.40	63.00	62.65	64.01
29	O_issue_opened	73.00	67.33	64.60	64.30	63.55	66.56
30	O_pr_assigned	44.00	29.33	26.60	20.50	21.30	28.35
31	O_pr_commented	48.00	45.67	47.80	48.00	50.15	47.92
32	O_pr_merged	74.00	55.67	50.40	45.00	47.40	54.49
33	O_pr_opened	70.00	63.33	61.80	61.10	62.00	63.65
34	pr_2_comment	58.00	53.33	54.80	56.60	52.95	55.14
35	pr_assigned	44.00	29.33	26.60	20.50	21.30	28.35
36	pr_commented	49.00	45.33	45.80	48.50	49.80	47.69
37	pr_merged	72.00	54.00	49.40	44.50	47.45	53.47
38	pr_opened	67.00	64.00	61.20	60.40	60.95	62.71
39	pr_related	63.00	62.33	62.80	64.00	63.70	63.17
40	Sun_metric	73.00	74.33	74.00	70.70	66.95	71.80

Table A4. All developer metrics all scores to evaluate with the experience approach (context-based).

ID	Developer Metrics (experience approach)	Context-based similarity					
		top1	top3	top5	top10	top20	mean
1	binary_code_contributions	76.00	73.33	73.40	74.30	72.15	73.84
2	binary_comments	75.00	72.00	74.40	75.20	73.20	73.96
3	binary_commit_related	70.00	67.67	69.20	65.90	64.45	67.44
4	binary_issue_related	75.00	71.67	72.60	75.00	73.50	73.55
5	binary_pr_related	78.00	78.67	77.40	74.40	72.25	76.14
6	code_2_comment	65.00	66.67	70.20	68.90	67.00	67.55
7	code_contributions	70.00	71.33	70.40	70.80	69.20	70.35
8	comment_2_code	60.00	68.33	67.60	67.40	66.95	66.06
9	comments	76.00	73.00	72.00	70.60	68.50	72.02
10	commit_2_comment	55.00	49.00	48.80	47.40	44.80	49.00
11	commit_authored	65.00	57.67	58.80	57.00	53.55	58.40
12	commit_commented	72.00	66.67	66.60	63.90	60.90	66.01
13	commit_committed	65.00	57.67	57.60	55.70	52.50	57.69
14	commit_related	72.00	67.67	67.60	64.50	63.55	67.06
15	issue_2_comment	66.00	68.67	70.80	70.50	68.75	68.94
16	issue_assigned	59.00	58.67	57.80	53.30	51.95	56.14
17	issue_closed	50.00	50.00	49.40	46.40	43.75	47.91
18	issue_hasPR	73.00	71.67	70.80	69.00	65.95	70.08
19	issue_commented	75.00	71.00	70.60	70.20	68.95	71.15
20	issue_opened	75.00	71.67	71.80	70.70	68.05	71.44
21	issue_related	76.00	73.00	71.40	69.60	68.70	71.74
22	O_commit_authored	61.00	57.00	57.40	53.10	51.15	55.93
23	O_commit_commented	65.00	69.00	67.60	65.20	61.45	65.65
24	O_commit_committed	60.00	56.00	54.80	51.30	50.05	54.43
25	O_issue_assigned	55.00	54.67	51.60	50.10	47.65	51.80
26	O_issue_closed	50.00	44.00	43.00	41.30	40.70	43.80
27	O_issue_closedPR	60.00	56.33	56.60	57.20	55.80	57.19
28	O_issue_commented	58.00	58.67	59.20	59.10	58.60	58.71
29	O_issue_opened	55.00	59.00	58.40	61.20	60.25	58.77
30	O_pr_assigned	36.00	31.33	30.60	30.00	28.90	31.37
31	O_pr_commented	62.00	65.33	64.20	61.80	59.00	62.47
32	O_pr_merged	60.00	57.67	57.80	57.90	55.50	57.77
33	O_pr_opened	55.00	54.67	53.00	54.60	55.75	54.60
34	pr_2_comment	58.00	58.00	56.40	54.60	53.20	56.04
35	pr_assigned	36.00	30.67	30.00	29.80	29.50	31.19
36	pr_commented	60.00	62.67	61.40	60.10	58.00	60.43
37	pr_merged	65.00	64.33	62.20	60.30	57.55	61.88
38	pr_opened	70.00	71.33	69.20	68.30	65.85	68.94
39	pr_related	69.00	71.00	71.20	67.60	67.15	69.19
40	Sun_metric	68.00	71.33	71.40	72.00	69.75	70.50

References

1. De Lima, M.L.; Soares, D.M.; Plastino, A.; Murta, L. Developers assignment for analyzing pull requests. In *Proceedings of the ACM Symposium on Applied Computing*; Association for Computing Machinery: New York, NY, USA, 2015; pp. 1567–1572. [\[CrossRef\]](#)
2. Badashian, A.S.; Hindle, A.; Stroulia, E. Crowdsourced bug triaging. In *Proceedings of the 2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015—Proceedings*, Bremen, Germany, 27 September–3 October 2015; pp. 506–510. [\[CrossRef\]](#)
3. Júnior, M.L.D.L.; Soares, D.M.; Plastino, A.; Murta, L. Automatic assignment of integrators to pull requests: The importance of selecting appropriate attributes. *J. Syst. Softw.* **2018**, *144*, 181–196. [\[CrossRef\]](#)
4. Zhang, L.; Zou, Y.; Xie, B.; Zhu, Z. Recommending relevant projects via user behaviour: An exploratory study on Github. In *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies, CrowdSoft 2014—Proceedings*, Hong Kong, China, 17 November 2014; Association for Computing Machinery, Inc.: New York, NY, USA, 2014; pp. 25–30. [\[CrossRef\]](#)
5. Sun, X.; Xu, W.; Xia, X.; Chen, X.; Li, B. Personalized project recommendation on GitHub. *Sci. China Inf. Sci.* **2018**, *61*, 1–14. [\[CrossRef\]](#)
6. Ozcan Kini, S.; Tosun, A. Periodic developer metrics in software defect prediction. In *Proceedings of the 18th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2018*, Madrid, Spain, 23–24 September 2018; pp. 72–81. [\[CrossRef\]](#)
7. McMillan, C.; Grechanik, M.; Poshyvanyk, D. Detecting similar software applications. In *Proceedings of the International Conference on Software Engineering, Zurich, Switzerland, 2–9 June 2012*; pp. 364–374. [\[CrossRef\]](#)
8. Hu, J.; Sun, X.; Lo, D.; Li, B. Modeling the evolution of development topics using Dynamic Topic Models. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015—Proceedings*, Montreal, QC, Canada, 2–6 March 2015; pp. 3–12. [\[CrossRef\]](#)
9. Dabbish, L.; Stuart, C.; Tsay, J.; Herbsleb, J. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW, Bellevue, DC, USA, 11–15 February 2012*; ACM Press: New York, NY, USA, 2012; pp. 1277–1286. [\[CrossRef\]](#)
10. Alarcon, G.M.; Gibson, A.M.; Walter, C.; Gamble, R.F.; Ryan, T.J.; Jessup, S.A.; Boyd, B.E.; Capiola, A. Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation. *Systems* **2020**, *8*, 28. [\[CrossRef\]](#)
11. Gousios, G.; Pinzger, M.; Deursen, A.V. An exploratory study of the pull-based software development model. In *Proceedings of the International Conference on Software Engineering, Shanghai, China, 18–21 August 2014*; IEEE Computer Society: New York, NY, USA, 2014; pp. 345–355. [\[CrossRef\]](#)
12. Thongtanunam, P.; Tantithamthavorn, C.; Kula, R.G.; Yoshida, N.; Iida, H.; Matsumoto, K.I. Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. In *Proceedings of the 2015 IEEE 22nd International on Software Analysis, Evolution, and Reengineering, SANER 2015—Proceedings*, Montreal, QC, Canada, 2–6 March 2015; pp. 141–150. [\[CrossRef\]](#)
13. Tsay, J.; Dabbish, L.; Herbsleb, J. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the International Conference on Software Engineering, Shanghai, China, 18–21 August 2014*; pp. 356–366. [\[CrossRef\]](#)
14. Yu, Y.; Wang, H.; Yin, G.; Ling, C.X. Reviewer recommender of pull-requests in GitHub. In *Proceedings of the 30th International Conference on Software Maintenance and Evolution, ICSME 2014, Riva del Garda, Italy, 23–30 September 2014*; pp. 609–612. [\[CrossRef\]](#)
15. Van Der Veen, E.; Gousios, G.; Zaidman, A. Automatically prioritizing pull requests. In *Proceedings of the IEEE International Working Conference on Mining Software Repositories, Florence, Italy, 16–17 May 2015*; pp. 357–361. [\[CrossRef\]](#)
16. Cosentino, V.; Izquierdo, J.L.C.; Cabot, J. Three Metrics to Explore the Openness of GitHub projects. *arXiv* **2014**, arXiv:1409.4253.
17. Kaur, G.; Bahl, K. Software Reliability, Metrics, Reliability Improvement Using Agile Process. *Int. J. Innov. Sci. Eng. Technol.* **2014**, *1*, 143–147.
18. Tiwari, V.; Pandey, R. Open source software and reliability metrics. *Int. J. Adv. Res. Comput. Commun. Eng.* **2012**, *1*, 808–815.
19. Bird, C.; Nagappan, N.; Murphy, B.; Gall, H.; Devanbu, P. Don't touch my code! Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering*; ACM Press: New York, NY, USA, 2011; pp. 4–14. [\[CrossRef\]](#)
20. Munson, J.C.; Elbaum, S.G. Code churn: A measure for estimating the impact of code change. In *Proceedings of the International Conference on Software Maintenance, Bethesda, MD, USA, 16–19 November 1998*; pp. 24–31. [\[CrossRef\]](#)
21. Foucault, M.; Teyton, C.; Lo, D.; Blanc, X.; Falleri, J.R. On the usefulness of ownership metrics in open-source software projects. In *Information and Software Technology*; Elsevier: Amsterdam, The Netherlands, 2015; Volume 64, pp. 102–112. [\[CrossRef\]](#)
22. Happel, H.J.; Maalej, W. Potentials and challenges of recommendation systems for software development. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Atlanta, GA, USA, 9–14 November 2008*; ACM Press: New York, NY, USA, 2008; pp. 11–15. [\[CrossRef\]](#)
23. Robillard, M.; Walker, R.; Zimmermann, T. Recommendation systems for software engineering. *IEEE Softw.* **2010**, *27*, 80–86. [\[CrossRef\]](#)
24. Sharma, L.; Gera, A. A Survey of Recommendation System: Research Challenges. *Int. J. Eng. Trends Technol.* **2013**, *4*, 1989–1992.

25. Nielek, R.; Jarczyk, O.; Pawlak, K.; Bukowski, L.; Bartusiak, R.; Wierzbicki, A. Choose a Job You Love: Predicting Choices of GitHub Developers. In Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, WI 2016, Omaha, NE, USA, 13–16 October 2016; Institute of Electrical and Electronics Engineers Inc.: New York, NY, USA, 2017; pp. 200–207. [[CrossRef](#)]
26. Casalnuovo, C.; Vasilescu, B.; Devanbu, P.; Filkov, V. Developer On boarding in GitHub: The role of prior social links and language experience. In Proceedings of the 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015—Proceedings, Bergamo, Italy, 30 August–4 September 2015; Association for Computing Machinery, Inc.: New York, NY, USA, 2015; pp. 817–828. [[CrossRef](#)]
27. Liu, C.; Yang, D.; Zhang, X.; Ray, B.; Rahman, M.M. Recommending GitHub Projects for Developer Onboarding. *IEEE Access* **2018**, *6*, 52082–52094. [[CrossRef](#)]
28. Onoue, S.; Hata, H.; Matsumoto, K.I. A study of the characteristics of developers’ activities in GitHub. In Proceedings of the Asia-Pacific Software Engineering Conference, APSEC, Bangkok, Thailand, 2–5 December 2013; IEEE Computer Society: Piscataway, NJ, USA, 2013; Volume 2, pp. 7–12. [[CrossRef](#)]
29. Murgia, A.; Concas, G.; Tonelli, R.; Ortu, M.; Demeyer, S.; Marchesi, M. On the influence of maintenance activity types on the issue resolution time. In *ACM International Conference Proceeding Series*; Association for Computing Machinery: New York, NY, USA, 2014; pp. 12–21. [[CrossRef](#)]
30. Jarczyk, O.; Jaroszewicz, S.; Wierzbicki, A.; Pawlak, K.; Jankowski-Lorek, M. Surgical teams on GitHub: Modeling performance of GitHub project development processes. *Inf. Softw. Technol.* **2018**, *100*, 32–46. [[CrossRef](#)]
31. Soares, D.M.; de Lima Júnior, M.L.; Plastino, A.; Murta, L. What factors influence the reviewer assignment to pull requests? *Inf. Softw. Technol.* **2018**, *98*, 32–43. [[CrossRef](#)]
32. Yu, Y.; Wang, H.; Yin, G.; Wang, T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Inf. Softw. Technol.* **2016**, *74*, 204–218. [[CrossRef](#)]
33. Niu, J.; Wang, L.; Liu, X.; Yu, S. FUIR: Fusing user and item information to deal with data sparsity by using side information in recommendation systems. *J. Netw. Comput. Appl.* **2016**, *70*, 41–50. [[CrossRef](#)]
34. Guo, G. Resolving data sparsity and cold start in recommender systems. In Proceedings of the 20th international conference on User Modeling, Adaptation, and Personalization, Montreal, QC, Canada, 16–20 July 2012; Volume 7379 LNCS, pp. 361–364. [[CrossRef](#)]
35. Şeker, A.; Diri, B.; Arslan, H.; Amasyali, F. Summarising big data: Public GitHub dataset for software engineering challenges. *Cumhur. Sci. J.* **2020**, *41*, 720–724. [[CrossRef](#)]
36. Karabadjji, N.E.I.; Beldjoudi, S.; Seridi, H.; Aridhi, S.; Dhifli, W. Improving memory-based user collaborative filtering with evolutionary multi-objective optimization. *Expert Syst. Appl.* **2018**, *98*, 153–165. [[CrossRef](#)]
37. Piantadosi, S.T. Zipf’s word frequency law in natural language: A critical review and future directions. *Psychon. Bull. Rev.* **2014**, *21*, 1112–1130. [[CrossRef](#)] [[PubMed](#)]
38. Jiang, J.; Lo, D.; He, J.; Xia, X.; Kochhar, P.S.; Zhang, L. Why and how developers fork what from whom in GitHub. *Empir. Softw. Eng.* **2017**, *22*, 547–578. [[CrossRef](#)]
39. Sheoran, J.; Blincoe, K.; Kalliamvakou, E.; Damian, D.; Ell, J. Understanding “watchers” on GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014—Proceedings, Hyderabad, India, 31 May–1 June 2014; Association for Computing Machinery, Inc.: New York, NY, USA, 2014; pp. 336–339. [[CrossRef](#)]
40. Kalliamvakou, E.; Gousios, G.; Blincoe, K.; Singer, L.; German, D.M.; Damian, D. An in-depth study of the promises and perils of mining GitHub. *Empir. Softw. Eng.* **2016**, *21*, 2035–2071. [[CrossRef](#)]