Zhang L, Zou Y, Xie B, et al. Recommending relevant projects via user behaviour: An exploratory study on github[C]//Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies. ACM, 2014: 25-30.

Recommending Relevant Projects via User Behaviour: An Exploratory Study on Github

Lingxiao Zhang, Yanzhen Zou, Bing Xie[*] , Zixiao Zhu
Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, P.R. China
{zhanglx12, xiebing, zouyz, zhuzx11}@sei.pku.edu.cn

本文旨在通过开发者之间的关联为开发人员推荐开源项目，进而快速创建原型

ABSTRACT

Social coding sites (e.g., Github) provide various features like Forking and Sending Pull-requests to support crowd-based software engineering. When using these features, a large amount of user behavior data is recorded. User behavior data can reflect developers preferences and interests in software development activities. Online service providers in many fields have been using user behavior data to discover user preferences and interests to achieve various purposes. In the field of software engineering however, there has been few studies in mining large amount of user behavior data. Our goal is to design an approach based on user behavior data, to recommend relevant open source projects to developers, which can be helpful in activities like searching for the right open source solutions to quickly build prototypes. In this paper, we explore the possibilities of such a method by conducting a set of experiments on selected data sets from Github. We find it a promising direction in mining projects' relevance from user behavior data. Our study also obtain some important issues that is worth considering in this method.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*Productivity*

General Terms

Experimentation

Keywords

Crowd-base software engineering, Social coding, Github, Recommendation system

[*]Corresponding author

CrowdSoft'14, November 17, 2014, Hong Kong, China
Copyright 2014 ACM 978-1-4503-3224-8/14/11...$15.00
http://dx.doi.org/10.1145/2666539.2666570

1. INTRODUCTION

群体化软件工程

Crowd-Based Software Engineering(CBSE) allows anyone to participate in software development tasks including documentation, design, coding and testing. CBSE is widely used in practise in recent years, especially in social coding sites like *Github*. Github encourages transparency and collaboration in software development activities[3]. It hosts software repositories on git, a distributed version control system, which supports *pull-based* development paradigm [4], allowing any developer to clone any public repository and commit changes at will. The maintainer of a project is able to pull valuable commits and branches to the original repository. By exploiting such software development paradigm, millions of free softwares emerge in the effort of community. Hosting more than 10 million repositories[1], Github has become the resource pool for many developers who are searching for open source solutions.

To support crowd-based software development, Github has implemented various features to improve the openness and dissemination of development process [7]. By using these features, developers can easily be notified of the latest changes of a project by "watching" a project, or clone a project's repository for his/her own use by "forking" a project. The use of these features generates abundant *user behavior data*, which contains rich information about the process of software development. To be specific, a developer only pays attention to *relevant projects* that satisfy certain preferences and interests. For example, a developer who is interested in data visualization may pay attention to projects written in R language that are suitable for statistical analysis and projects for data visualization like *d3.js*[2]; A web application developer who is a back-end expert may be interested in cache management systems like *Memcached*[3] or *Redis*[4]. User behavior data records developers' activities around relevant projects, thus can reflect user preferences and interests. As a result, if two projects are relevant in satisfying the same preference or interest, it is most likely that similar groups of people will pay attention to them, generating similar user behavior data. So it is possible for us to find relevance between such projects via user behavior.

Based on this hypothesis, it is a promising method to help developers find *relevant projects* by mining user behav-

[1]https://github.com/blog/1724-10-million-repositories
[2]https://github.com/mbostock/d3
[3]https://github.com/memcached/memcached
[4]https://github.com/antirez/redis

ior data. Provided that most of the developers nowadays build prototype applications based on multiple open source building blocks, recommending relevant projects(e.g., recommending *d3.js* to R programmers, or recommending *Redis* to a *Memcached* user) can be helpful to developers that are searching for the right open source components. In order to propose such a recommendation method, we need to find a way of properly describing relationships between projects in user data. It is also necessary to pay attention to pros and cons in such a method. In this paper, we explore the possibilities of finding relevant projects via analyzing user behavior data. Specifically, we try to answer three research questions below:

RQ1 What types of user behavior data are suitable for recommending relevant projects?

RQ2 What are the relationships between relevant projects found via user behavior?

RQ3 How do projectsąŕ types affect recommendation results?

Aiming at answering these questions, we design a naive vector similarity method to measure the relevance between projects. We perform our method on a selected data set from Github to generate a series of results. The details of our research design is presented in Section 2. We use statistical analysis combined with manual labelling to draw answers to these questions. Our findings are listed in Section 3. We discuss related work and give our conclusion in the remaining sections.

2. RESEARCH DESIGN

To answer our research questions, we select multiple user behavior data sets as our data source. An exploratory method is performed to find relevant projects on each data sets.

2.1 Data

The data sets we use[5] are contributed by the work of Georgios Gousios and Diomidis Spinellis [5]. It includes data of 89 most starred Github projects of 9 most commonly used programming languages. The number of Github users involved in this data set is 499,485. The time of the recorded user behavior data ranges from 2008 when Github is launched, to September of 2013. From the various features implemented on Github, we select 5 most commonly used ones as our user behavior data source. We describe each of these features below:

Fork. To fork a project means to clone the entire git repository of the project. After forking, a developer owns a copy of the original repository and gains full access to any version control data. Forking is usually the first step to start using a project or to create a new branch to make contribution to the original project.

Watch. Watching a project is like following a person on twitter, which means subscribing every event that happens to this project, including branch merging, version evolving and bug reporting.

Issues Comment. Github has its own issue tracking system. A public project's issue tracking system is transparent to all users on Github, which means everyone is free to raise or discuss issues. For convenience we do not discriminate users who raise issues from those who comment on others' issues. They are considered equivalent in issue discussions.

[5]http://ghtorrent.org/msr14.html

Table 1: Number of Records in Data Set

Data Set	Number of Records
Fork	108,628
Watch	295,798
Comment on issues	534,104
Pull-Request	78,955
Membership	1,941

Pull-Request. If a developer wants to contribute to a project, but is not an authorized committer, he or she can send a *Pull-Request* to the project which includes the code commits. The maintainers of the project are then able to review the code to decide whether the pull-request can be added to the main branch.

Member. To gain membership of the project means to become an authorized committer and to have direct access to commit to the central repository.

We exclude some other important features provided by Github like following other users, we do not use developer-developer relationship in our method at the moment. The 5 features we choose give us 5 different data sets, listed in Table 1. Each of the data sets records feature usage data about the *user* and the *target project*, i.e., when a user forks(or watches/adds a comment to the issues of/sends a pull-request to/become the member of) a project, a new record is added to the corresponding data set. It is to be noted that some of these features are open to all users on Github like *Fork* or *Watch*, others may require some professional skills and knowledge like *Pull-Request* or *Member*.

2.2 Method

The basic idea of our method is to measure the relevance between two projects by comparing how similar the user behavior data on two projects is. Our method consists of the following steps:

1st. For each of the 5 features, represent a project as an n-dimensional *user vector* with n as the Number of all users. Take *Fork* as an example:

$$\vec{P}_{Fork}(i) = (u_1, ..., u_n)' \qquad (1)$$

Each u_i is either 0 or 1 depending on the records in the data sets. For example, if user k has forked project j, then u_k in $\vec{P}_{Fork}(j)$ is 1. With the data of 5 features, we now have 5 types of user vectors for each project.

2nd. For each type of user vectors, calculate pairwise *relevance score* for all projects using cosine distance. For instance, the *Fork* relevance score of Project i and Project j is calculated as such:

$$\begin{aligned} RelScore_{Fork}(i,j) &= cos(\vec{P}_{Fork}(i), \vec{P}_{Fork}(j)) \\ &= \frac{\vec{P}_{Fork}(i) \bullet \vec{P}_{Fork}(j)}{\|\vec{P}_{Fork}(i)\| * \|\vec{P}_{Fork}(j)\|} \end{aligned} \qquad (2)$$

3rd. For each project, rank other projects by relevance score. As a result, for each project, our method generates 5 ranked lists of 88 other projects using 5 types of user behavior data sets.

To answer the research questions, we perform quantitative methods combined with qualitative analysis on the generated results. For quantitative analysis, we focus on the statistical characteristics of the relevance scores in the ranked

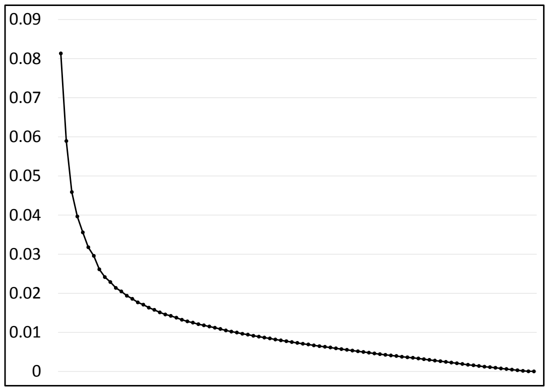

Figure 1: Average Relevance Score List of Fork

lists. For qualitative analysis, we manually label the relationships between the relevant projects found by our method into 4 types(see Section 3.2). We recruit 3 graduate students in computer science major with more than 4 years of programming skills to label 1283 pairs of relationships. We divide the 1283 items into 3 subsets with each student labelling 2 subsets for cross validation. Conflicts in labelling is resolved by group discussion. More details our analysis methods are presented in Section 3.

3. STUDY RESULT

In this section, we present statistic results to answer all research questions. We also use manual labelling to answer RQ2. The detail of the studies for each question is described below.

3.1 What types of user behavior data are suitable for recommending relevant projects?

To answer RQ1, we need to inspect the statistical characteristics of the generated ranked lists. As mentioned in Section 2.2, we have generated 5 ranked lists for each project, that is, we have 5 sets of lists for 5 user behavior data sets, each containing 89 ranked lists of 89 projects. Each ranked list has 88 items with the corresponding relevance scores. For each set of lists, we generate a *Average Relevance Score List*, by calculating the average of the relevance scores at each position for all 89 projects. As a result, we have 5 average relevance score lists, representing the statistical characteristics for 5 types of user behavior data.

Figure 1 shows the average score list generated by records of *Fork*. The y-axis is the value of the average relevance score, the x-axis the ranked sequence number The shape of the chart clearly shows that the scores of top results significantly exceed the rest, which are a long tail of low-value results. The chart indicates that a project is significantly relevant with a few other projects, and has low relevance score with most of other projects. This indication agrees with the common sense that developers are only interested in projects that satisfy their preferences, thus a project can only be relevant to a few other projects that are related to a common preference or purpose. The chart of *Fork*, as a result, shows that the user behavior data of *Fork* is able to reflect user preferences and interests. We conclude that the data set of *Fork* is suitable for recommending relevant projects.

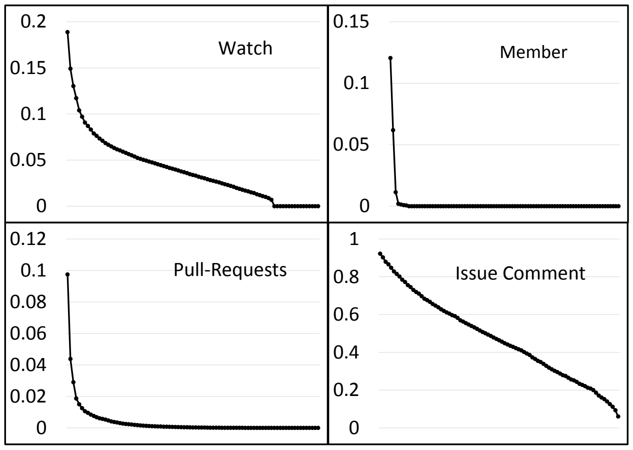

Figure 2: Average Relevance Score Lists of 4 Other Data Sets

We present the average lists generated by the other 4 user behavior data sets in Figure 2. The results of *Watch, Pull-Request* and *Member* are all similar to the result of *Fork*, with top results of high relevance scores and a long tail of low relevance scores. As our previously analysis, the 3 types of data are all suitable for recommending relevant projects.On the other hand, the result generated by *Issue Comment* is entirely different. The shape of the chart does not show a significant difference between top results and bottom results. instead, it presents a linear relationship between the relevance scores and the ranked sequence number. We would have a similar ranked list if each relevance score is randomly given within a certain range. In a word, the result of *Issue Comment* data set resembles that of a randomly generated data set and contains no valuable information regarding relevance between projects. From the results presented above, we give our first finding below:

Finding 1. *4 types of user behavior data sets including* Fork*,* Watch*,* Pull-Request *and* Member *are suitable for recommending relevant projects, while the* Issue Comment *data set is not suitable in our method and needs further study.*

In the studies afterwards, we exclude the results generated by *Issue Comment* data set based on this finding.

3.2 What are the relationships between relevant projects found via user behavior?

One important issue in building a recommendation system is the explanation of the recommended results. In our method, we need to explain why top results in the ranked list generated by user behavior data are *relevant* to a certain project. We observe top results of the generated ranked lists and find 3 major types of relationship between projects. Most of the relationships between two *relevant* projects belong to the 3 types listed below:

Dependence. Project A is dependent on project B if B is used in A as a dependent library, a framework or other necessary components. For example, Project *Nodejs*[6] is dependent on Project *Libuv*[7] because it is build on top of *Libuv's* asychronous I/O library.

[6]https://github.com/joyent/node
[7]https://github.com/joyent/libuv

Table 2: Distribution of Relationships in Top-1 Results

Top-1

Data set	Dependence	Co-use	Similar	Unknown
Fork	37.93%	40.23%	**14.94%**	6.90%
Watch	21.13%	**49.30%**	**16.90%**	12.68%
Pull-req	**45.35%**	37.21%	4.65%	12.79%
Member	**48.65%**	35.14%	5.41%	10.81%
Total	37.37%	40.93%	11.03%	10.68%

Top-5

Data set	Dependence	Co-use	Similar	Unknown
Fork	19.08%	**39.08%**	**7.82%**	34.02%
Watch	15.30%	**36.26%**	**8.22%**	40.23%
Pull-req	20.91%	32.45%	5.29%	41.35%
Member	**41.77%**	26.58%	2.53%	29.11%
Total	20.03%	35.39%	6.78%	37.80%

Co-use. This type of relationship indicates that two projects are often used together to serve a common purpose. R-language-based projects are often used with *Storm*[8], a distributed computation framework, to achieve mass scale data processing.

Similar. This means two projects have similar features or functions, like *Memcached* and *Redis*, which can both be used as memory caching systems.

Each of 3 types of relationship serves different recommendation purposes. For example, a project contributor is more interested in what the project is dependent on, but an ordinary user of the project is more interested in projects that can be co-used or with similar features. To determine what recommendation purposes each user behavior data sets are suitable for, we manually label top 5 results of all ranked lists generated by 5 data sets. The 3 students we hired label the results into 4 types including the aforementioned 3 and an *Unknown* type, following this instruction: Firstly decide if two projects are *similar* by reading the introduction in the projects home page; If not, perform code search on two projects to verify *dependency*; At last, use both projects' name as keywords to conduct a search on *Google*, browsing through the results on the first page to see if Co-use exists; If the above steps fail to find a meaningful relationship, then label it as *Unknown.* The results are presented in Table 2. It is to be noted that we neglects the results with a relevance score of zero when labelling.

We labelled 281 items in top-1 results in the ranked lists. The total results shows that approximately 90% of Top-1 listed items are identified to have one of the 3 types of relationships with the corresponding projects. We also present the distribution for the top-1 results generated by each of 4 types of user behavior data. We can see that different user behavior data contribute differently to each type of relationship. The bold-font results show the percentages which significantly exceed the corresponding percentages in total result. We can infer that *Fork* and *Watch* may be used more by ordinary users, for the data sets generate more results with relationships of *Co-use* and *Similar*, which ordinary users are likely to care about. On the contrary, the data sets of *Pull-Request* and *Member* are more suitable for finding relationships of *Dependence.* For top-5 results in the ranked lists we labelled 1283 items. Comparing with the top-1 results, the percentage of *Unknown* relationship significantly increased, for it is harder to find meaningful relationships between projects with low relevance scores. For the distributions generated by each data sets, comparing to the top-1 results, we can still see the similar differences in percentages contributed by different data sets. Thus, for RQ2, we have our conclusion below:

[8]https://github.com/nathanmarz/storm

Table 3: Project Classification

Category	Description	Number of Projects
User App	Projects like a web site or a mobile apps that can be used by non-programmers	17
Library	A collection of useful implementations that can be referenced in user programs	22
Component	Stand-alone functional parts that run independently and provide APIs to other programs e.g., a database	10
Framework	Reusable software platforms that are selectively changeable by additional user-written codes for user-specific applications	23
Dev-tools	Basic tools supporting software development like programming language, runtimes, package managing tools.	11

Finding 2. *Most of relevant projects found by our method can be identified to have one of 3 types of relationships which are Dependence, Co-use and Similar. The data sets of Fork and Watch are more suitable to find Co-use and Similar relationships, and the data sets of Pull-Request and Member are more suitable for Dependence relationships.*

3.3 How do projects' types affect recommendation results?

When browsing through the top results of the ranked lists, we notice a certain diversity in the recommendation effectiveness between different projects. More specifically, we find that for most projects, their relationships with the recommended relevant items can be explained in one of 3 types labelled in Section 3.2, but for some projects, most of their recommended relevant items are labelled as *Unknown.* For these projects, our method shows low recommendation effectiveness, regardless of the type of user behavior data set we use. Based on this observation, we suspect that a project's type has a certain impact on the recommendation results.

To confirm our suspicion, we classify 89 projects into 5 categories to see the statistical characteristics of the ranked lists for each type of projects. We list the classification details in Table 3. We exclude 5 projects that can not be classified into any of the 5 categories, including 3 CMS[9] projects

[9]http://en.wikipedia.org/wiki/Content_management_system

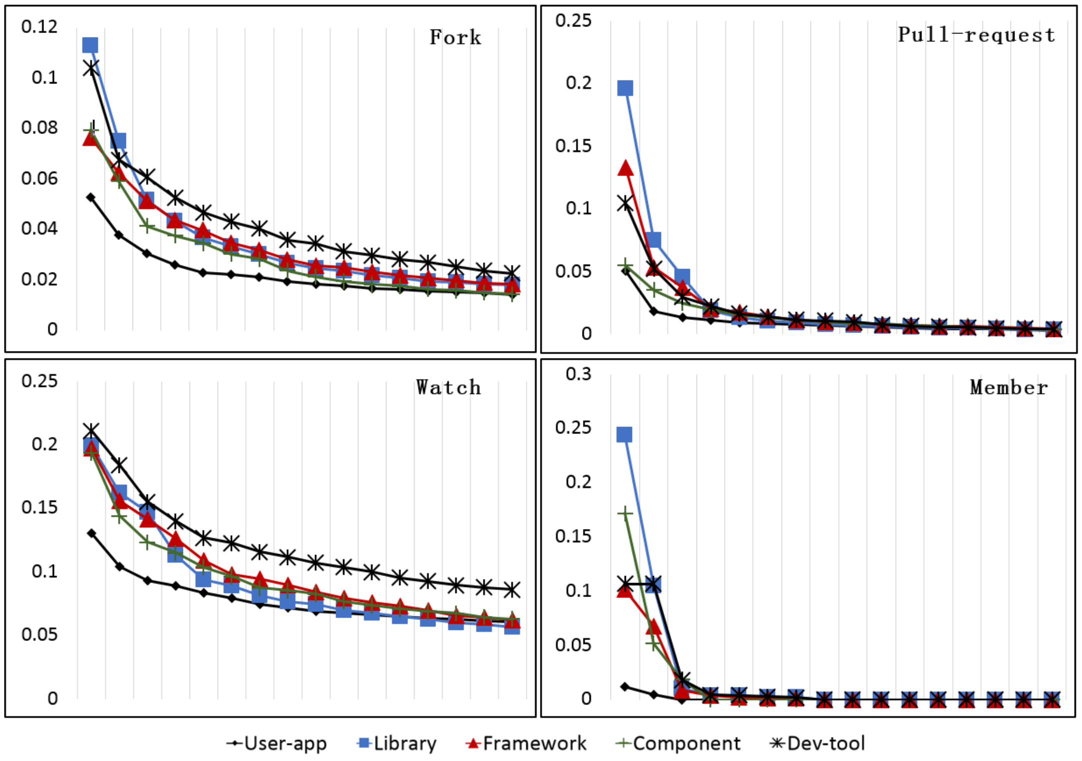

Figure 3: Average Relevance Score Lists of Each Category

and 2 template projects[10]. For projects in each category, we use the method in Sectionrefsec:rq1 to generate the *Average Relevance Score Lists* of 4 user behavior data sets. As a result, for each user behavior data set, we have 5 *Average Relevance Score Lists* of each project category. We present the results in Figure 3.

As analyzed in Section 3.1, only top items in each ranked list are closely relevant to the corresponding projects, so we focus on the differences between each category in top items in the average lists. From the figure we can identify 2 mutual characteristics of the 4 charts. **First**, the average relevance scores of *User App*'s relevant projects are significantly lower than those of other categories. **Second**, except for the chart of *Watch*, in which the results of each category are similar, the average relevance scores of *Library*'s relevant projects are above those of other categories. These findings can be viewed as evidences of different user behavior paradigm on different categories of projects. On one hand, *User Apps* can attract all kinds of users for they require few professional knowledge in programming, a user of such a project is not necessarily an expert of the related field. As a result, the user behavior data of such projects contains few information regarding users' preferences and interests, which leads to poor recommendation results. On the other hand, to use a *Library* means to have enough professional knowledge of the programming language as well as the library's API documentation. Most of the a *Library's* users are programmers with a certain level of expertise in the related field. Thus, the user behavior data of such projects can properly reveal users' preferences and interests, making the recommendation results more accurate. To conclude our study on RQ3, a project's type can affect the effectiveness of our method:

Finding 3. *User behavior data is more effective against Library category in recommending relevant projects, and may have poor results on User App projects.*

4. RELATED WORK

We propose a method suitable for building a recommendation system. Recommendation System for Software Engineering(RSSE) is an important field of Research. Robillard et al. [11], Happel and Maalej [6] has summarized important issues and some representative work. Most researcher in this field focus on recommending software artifacts of small grain such as code snippets or bug reports. One that is similar with ours in recommendation grain is the work by McMillan et al [10], who detect similar applications by analyzing package structure. Our work focus on find a broader sense of "relevant" projects other than similar ones. The data source we use is also different.

The data sets used by our method are from Github, which attracts much academic attention in recent years. The shift of development paradigm promoted by Github has been discussed by many researchers including Dabbish et al. [3], Begel et al. [1], McDonald and Goggins. [9]. The data sets provided by github have also become the resource pool for many researchers. Bird et al. [2] summarized promises and perils of mining Github data sets. We have not found any research work on recommending projects from Github, but there have been many studies on mining Github data sets to enhance software engineering from other aspects. Mar-

[10]See https://github.com/johnmyleswhite/ProjectTemplate and https://github.com/h5bp/html5-boilerplate

low et al. [8] use user behavior data to form activity traces and personal profiles. Vasilescu et al. [13] establish associations between Github users and StackOverflow users to find relationships between user behaviors on the two web sites. Thung et al. [12] build network structure of repositories and users on Github to study influence of important projects and developers. Gousios et al. [4] use Github data sets to study the mechanism of pull-based development and to help maintainers to decide pull-requests' quality.

5. CONCLUSION

In this paper, we conduct an exploratory study on finding relevant projects via user behavior. We design a simple method of mining relevant projects from user behavior data sets from Github. In our experiments, we explore the possibility of such a method and other related issues. From the results we draw 3 major findings: Firstly, 4 types of user behavior data including *Fork*, *Watch*, *Pull-Request* and *Member* are suitable for finding relevant projects, while the *Issue Comment* data set is not suitable in our method; Secondly, different user behavior data sets are suitable for different recommendation purposes; Thirdly, projects' type can affect the effectiveness of our method. Base on current findings, we conclude that it is a promising method to mine relevance between projects in user behavior data. In the future, we hope to extend this method into a mature approach to build a real-world project recommendation system.

6. ACKNOWLEDGMENT

This work is supported by the High-Tech Research and Development Program of China under Grant No. 2012AA011202; the National Basic Research Program of China (973) under Grant No. 2011CB302604, the National Natural Science Foundation of China under Grant Nos. 61103024, U1201252.

7. REFERENCES

[1] A. Begel, J. Bosch, and M.-A. Storey. Social networking meets software development: Perspectives from github, msdn, stack exchange, and topcoder. *Software, IEEE*, 30(1):52–66, 2013.

[2] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. The promises and perils of mining git. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, pages 1–10. IEEE, 2009.

[3] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1277–1286. ACM, 2012.

[4] G. Gousios, M. Pinzger, and A. van Deursen. An exploratory study of the pull-based software development model. In *Software Engineering (ICSE), 2014 36th International Conference on*, pages 345–355. IEEE, 2014.

[5] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21. IEEE, 2012.

[6] H.-J. Happel and W. Maalej. Potentials and challenges of recommendation systems for software development. In *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*, pages 11–15. ACM, 2008.

[7] J. Jiang, L. Zhang, and L. Li. Understanding project dissemination on a social coding site. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 132–141. IEEE, 2013.

[8] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: activity traces and personal profiles in github. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 117–128. ACM, 2013.

[9] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 139–144. ACM, 2013.

[10] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 364–374. IEEE, 2012.

[11] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *Software, IEEE*, 27(4):80–86, 2010.

[12] F. Thung, T. F. Bissyandé, D. Lo, and L. Jiang. Network structure of social coding in github. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 323–326. IEEE, 2013.

[13] B. Vasilescu, V. Filkov, and A. Serebrenik. Stackoverflow and github: associations between software development and crowdsourced knowledge. In *Social Computing (SocialCom), 2013 International Conference on*, pages 188–195. IEEE, 2013.