Recommending Open Source Software Projects to Developers

Shahab Bayati
Information Systems and Operations Management

University of Auckland School of Business
s.bayati@auckland.ac.nz

Arvind K Tripathi
Information Systems and Operations Management

University of Auckland School of Business
a.tripathi@auckland.ac.nz

Ravi Bapna
Information Systems and Operations Management

Carlson School of Management

University of Minnesota
rbapna@umn.edu

ABSTRACT

Growing popularity of OSS has attracted millions of developers to social coding platforms such as GitHub.com. However, it appears that OSS software is becoming a victim of its own success because finding the right project, among millions of projects hosted on social coding platforms, is a gruelling task for developers. Lack of mismatch among developers and projects has resulted in high developer turnover and project failures. In this context, the evolving nature of developers' preferences and projects' goals, complicates matching of developers and projects.

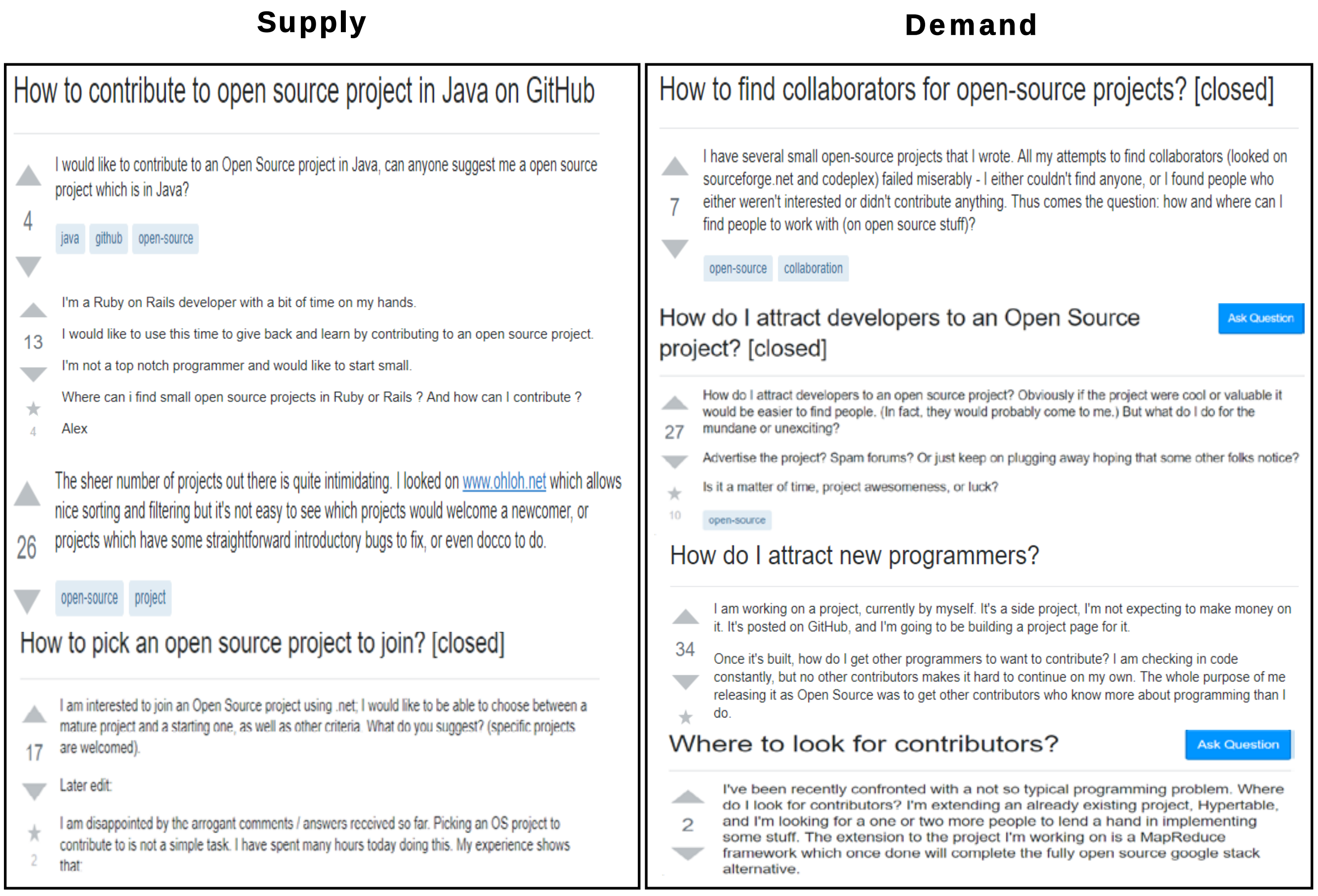
This paper proposes a new artefact based on collaborative filtering (CF) recommendation technique to recommend OSS projects to developers. The dynamic nature of projects' evolution and the developers preferences makes this a very different problem than, say, recommending products to consumers. Our proposed method uses developers' socio-technical activities to capture their evolving preferences and project goals, and creates an implicit personalized project rating/ranking for developers. A multi-criteria decision-making technique is used to generate an overall rating based on developers' different types of activities. The proposed artefact has been evaluated with the real-world data from GitHub. Our results show that developers who join projects that we recommend, are among the top contributors on these recommended projects, and vice versa for the developers who join projects that we don't recommend. The comparison of proposed method with other state of the art collaborative filtering approaches shows promising results.

INTODUCTION

Compared to its proprietary counterpart, Open Source Software (OSS) and applications are innovative, easier to use and have lower development cost (Paulson et al. 2004). Due to these qualities, OSS paradigm has gained tremendous popularity in recent years, even among traditional software companies such as Apple (Apple.com, 2015). Software firms who perceived OSS as a threat, have now changed their business model and started partnering with, and sponsoring OSS projects. Currently a large number of popular OSS projects are owned and sponsored by companies such as Facebook, Twitter, Uber, etc. This popularity growth and business interest has attracted scores of software developers to OSS ecosystem who wish to join and contribute to OSS projects hosted on social coding platforms such as GitHub.com, SourceForge.net, and Bitbucket.org. However, therein lies a challenge for the developers- how to find the right project to make contribution among millions of projects hosted on these social coding platforms?

Since OSS projects are a result of volunteer contributions, developers collaborate on these projects on their own terms. Therefore, social coding platforms, facilitate developers' collaborative preferences (preferences, hereafter) to maximize developers' '*collaborative utility*' (Blincoe, et al., 2016). Note that, developers' preferences (patterns) are observable from developers' socio-technical activities stored in OSS repositories. These patterns are diverse, representing diversity in developers' preferences and global nature of OSS teams (Dabbish, et. al., 2012). Since harmonious collaboration among team members are critical for success and productivity of OSS projects, we contend that considering developers' preferences are critical in matching them with OSS projects for long-term success of OSS ecosystem. OSS literature has argued that developers' collaborative network (captures *with whom they collaborate*) drive their project selection (Hann, et al., 2008). We add to this literature and argue that developers' preferences (captures *how they collaborate*) are also critical in their project selection.

GitHub one of the large social coding platforms, aims to act as a clearinghouse between OSS projects and developers, and facilitates a feature called *Explore GitHub*. However, *Explore GitHub* just allows searching for trending projects and is not a recommendation agent that can match developers and projects. We argue that the sheer size of these platforms makes is very difficult for both projects and developers to find a good match. As of 2019, GitHub has more than 50 million developers and 100 millions of projects. Figure 1 shows a few samples posts from a prominent online community forum Stackoverflow, highlighting how developers appeal for help to find appropriate projects and project owners look for OSS contributors.



How to contribute to open source project in Java on GitHub

4 — I would like to contribute to an Open Source project in Java, can anyone suggest me a open source project which is in Java?

13 — I'm a Ruby on Rails developer with a bit of time on my hands.

I would like to use this time to give back and learn by contributing to an open source project.

I'm not a top notch programmer and would like to start small.

Where can i find small open source projects in Ruby or Rails ? And how can I contribute ?

Alex

4

26 — The sheer number of projects out there is quite intimidating. I looked on www.ohloh.net which allows nice sorting and filtering but it's not easy to see which projects would welcome a newcomer, or projects which have some straightforward introductory bugs to fix, or even docco to do.

How to pick an open source project to join? [closed]

17 — I am interested to join an Open Source project using .net; I would like to be able to choose between a mature project and a starting one, as well as other criteria. What do you suggest? (specific projects are welcomed).

Later edit:

I am disappointed by the arrogant comments / answers received so far. Picking an OS project to contribute to is not a simple task. I have spent many hours today doing this. My experience shows that:

2

How to find collaborators for open-source projects? [closed]

7 — I have several small open-source projects that I wrote. All my attempts to find collaborators (looked on sourceforge.net and codeplex) failed miserably - I either couldn't find anyone, or I found people who either weren't interested or didn't contribute anything. Thus comes the question: how and where can I find people to work with (on open source stuff)?

How do I attract developers to an Open Source project? [closed]

27 — How do I attract developers to an open source project? Obviously if the project were cool or valuable it would be easier to find people. (In fact, they would probably come to me.) But what do I do for the mundane or unexciting?

Advertise the project? Spam forums? Or just keep on plugging away hoping that some other folks notice?

Is it a matter of time, project awesomeness, or luck?

10

How do I attract new programmers?

34 — I am working on a project, currently by myself. It's a side project, I'm not expecting to make money on it. It's posted on GitHub, and I'm going to be building a project page for it.

Once it's built, how do I get other programmers to want to contribute? I am checking in code constantly, but no other contributors makes it hard to continue on my own. The whole purpose of me releasing it as Open Source was to get other contributors who know more about programming than I do.

Where to look for contributors?

2 — I've been recently confronted with a not so typical programming problem. Where do I look for contributors? I'm extending an already existing project, Hypertable, and I'm looking for a one or two more people to lend a hand in implementing some stuff. The extension to the project I'm working on is a MapReduce framework which once done will complete the fully open source google stack alternative.

Figure1. Evidence from StackOverflow.com: Left panel: Developers seeking OSS projects; Right panel: OSS projects looking for contributors.

Since open source software powers large and small corporations alike, it plays a significant role in economic growth and therefore developers' sustained associations and contributions to these projects are very critical for their long-term success. Current state of art approaches recommend project selection based on project popularity (Jarczyk et al. 2014), license type (van Osch et al. 2011) and previous ties (Hahn et al. 2008; Jarczyk et al. 2014; van Osch et al. 2011) but are limited in many ways. For example, popular projects may attract more contributors (Fronchetti et al. 2019; Nielek et al. 2016), however, high turnover of OSS developers confirms that developers should evaluate projects with various metrics and not only the reputation (Bayati 2018). Developers joining projects based on prior ties (Hahn et al. 2008) is inefficient because it doesn't account for evolving nature of developers' preferences and project goals, and approach is not scalable for millions of developers and projects hosted on social coding platforms. Overall, these filtering mechanism aren't effective in OSS ecosystem due to its size and dynamic nature, resulting into low retention rates of developers (Schilling et al. 2012b). Recent studies confirm that a large number of developers leave OSS projects due to inappropriate project selection (Steinmacher et al. 2015) leading to project failure (Schilling 2012).

Scholars argue that a recommender system to recommend projects to developers is critical for long-term sustainability of OSS projects and communities (Jiang et al. 2016). Recommender systems are software applications that predict user preferences based on different approaches, such as Collaborative filtering (CF), content-based recommendation, knowledge base and hybrid systems (Ricci et al. 2011). Mining software repositories (MSR) techniques have been used to design recommender systems for software engineering (RSSE), but these are mainly focused on supporting software developers/practitioners and project managers in a variety of tasks such as Bug triaging, finding experts (Anvik et al. 2006; Anvik and Murphy 2011; Badashian et al. 2015; Naguib et al. 2013; Shokripour et al. 2013), source code suggestion (Holmes and Murphy 2005), etc. in the software development life cycle (SDLC) (Robillard et al. 2010), partly because recommending projects to developers is not trivial, as we discuss next.

Recommending OSS projects to developers is different from recommending products to consumers in online markets due to following challenges. First, OSS projects are not static, like other products, and evolve over time (Schafer et al. 1999; Wang et al. 2015). For example, OSS projects' content, associated developers, goals, organization, requirements, structure and governing policies change over time and influence developers' preferences. Second, developers themselves are different from online consumers. Over time developers' knowledge and skills change based on their contribution and engagement with OSS ecosystem, thereby changing their preferences. Developers' preferences also change with their experience. For example, experienced and skilled developers deserve and desire higher roles in projects (Robillard et al. 2014). Third, developers' rating for OSS projects are needed to design a collaborative filtering (CF) based recommender system. However, there is no mechanism on GitHub, or other social coding platforms to capture developers' ranking for OSS projects. Although, developers' watch list of projects can be used to construct their project rankings, but it'd not be helpful because- a) measurement is binary (1 if a project is on the watch list) and not ranked, b) watching doesn't signal developers' intent to contribute and c) developers' interests in OSS projects are temporal and dynamic, therefore, explicit rating would not be useful. In this research, we address all these three issues by developing a personalized project rankings for developers, which are used in a recommender system to recommend projects to developers.

Since there is no rating mechanism on GitHub to capture developers' interests in OSS projects, this study aims to design a new personalized project rating mechanism for developers. Using developers' socio-technical activities in OSS projects, openly available on GitHub, we employ implicit feedback (Kelly and Teevan 2003) approach to compute personalized project ratings. While many studies have used/developed implicit ratings in absence of explicit ratings from the users(Choi et al. 2012; Lee et al. 2010), however, none of them are comparable to our context where both developers' preferences and projects' goals are changing over time.

Developers are heterogeneous in their project selection and motivation for contribution. Scholars argue that multiple criterion such as socialization, learning new skills, gaining reputation and many other social and technical factors drive how developers select projects for contribution (Fang and Neufeld 2009; Roberts et al. 2006). Since project selection is a multi-criteria decision-making process, our artefact for personalize project rankings takes that into account. Multi-criteria recommender systems (MCRS) are studied in a wide range of domains and applications such as e-commerce and entertainment (Adomavicius and Kwon 2015; Palanivel and Sivakumar 2010). Compared to general recommender systems, MCRS consider various features of an item, rate them separately and aggregate later for an overall rating for the item. Since developers may have different and evolving preference for their socio-technical activities, we apply MCRS approach in our recommender system. Details are discussed in our artefact.

We follow Design Science Research (DSR) methodology (Hevner et al. 2004) and Peffers' model (Peffers et al. 2007) for designing this new artefact. Figure 2 shows our research phases in a schematic model. Figure 2 illustrates the performed phases in the first iteration to design a time-discounted implicit feedback rating technique. The main contributions of this research are following. First, we use developers' socio- technical activities, to implicitly rate these projects. Second, in creating the personalized project rankings, we account for the changes in developers' preferences and projects' goals over time, which is different from existing studies on OSS project selection (Allaho and Lee 2013; van Osch et al. 2011). Third, we consider the project selection as a multi-criteria decision-making (MCDM) process and therefore develop a multi-criteria recommender system (MCRS) to make a personalized project recommendation. Based on the abovementioned contributions the research question of this study is "How to design a multi-criteria recommender system when both consumers (developers) and products (projects) are evolving over time?”

This paper follows recommended structure for DSR studies (Gregor and Hevner 2013). The next section describes related work to OSS studies in information systems literature, the socio-technical analysis in OSS domain and RSSEs. Later we introduce the research method, which provides details of the proposed artefact followed by the evaluation process. Finally, we summarize the contributions and limitations of this study and provide directions for future research.

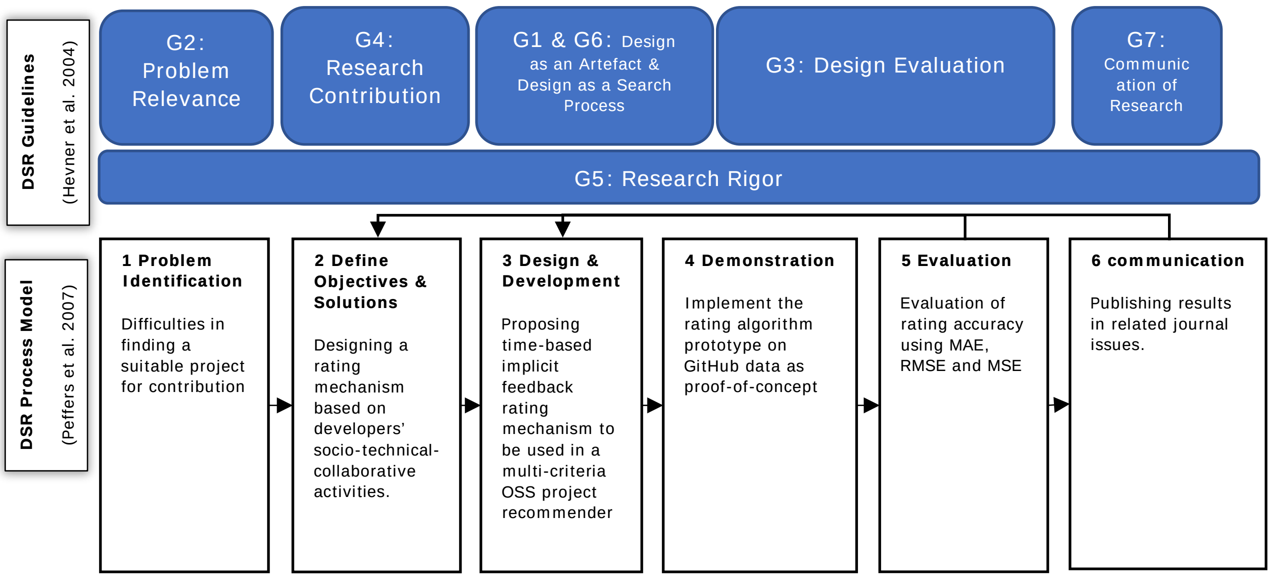

DSR Guidelines
(Hevner et al. 2004)
G2: Problem Relevance
G4: Research Contribution
G1 & G6: Design as an Artefact & Design as a Search Process
G3: Design Evaluation
G7: Communication of Research
G5: Research Rigor
DSR Process Model
(Peffers et al. 2007)
1 Problem Identification
Difficulties in finding a suitable project for contribution
2 Define Objectives & Solutions
Designing a rating mechanism based on developers' socio-technical-collaborative activities.
3 Design & Development
Proposing time-based implicit feedback rating mechanism to be used in a multi-criteria OSS project recommender
4 Demonstration
Implement the rating algorithm prototype on GitHub data as proof-of-concept
5 Evaluation
Evaluation of rating accuracy using MAE, RMSE and MSE
6 communication
Publishing results in related journal issues.

Figure 2. Research Model based on (Hevner et al. 2004) guidelines and (Peffers et al. 2007)Model

RELATED WORK

In this section, we discuss the literature on how developers join OSS projects, how OSS projects attract and retain developers and recommender systems in software engineering. We summarize these streams of literature and highlight our contribution.

Joining Open Source Projects

A large number of studies have established that developers are driven by a mix of intrinsic, extrinsic, ideological and community motivations (Alexy and Leitner 2011; Krishnamurthy et al. 2014; Von Krogh et al. 2003) to make contributions to OSS projects. However, we are yet to understand how these motivated developers select or join OSS projects. Since, developers' volunteer contributions are key to the success of OSS projects (Von Krogh et al. 2003), it is critical to expand our understanding on how developers find and join OSS projects. Note that matching developers' skills and preferences with project goals is key for sustainable contributions (Schilling et al. 2012b).

Scholars have examined developers' project selection process but these studies have mostly investigated the antecedents of joining a project. These studies find that the rate of code commit (Nielek et al. 2016), license type of projects (van Osch et al. 2011), required skillset (Terceiro et al. 2012), project reputation and popularity (Jarczyk et al., 2014; Fronchetti et al., 2019) , previous social ties and peer influence (Allaho and Lee 2013; Hahn et al. 2006) , and contribution of popular developers (Blincoe et al. 2016; Schilling et al. 2012b) are major drivers for developers' joing a project. While these studies highlight drivers of joining a project, they are thin on investigating the role of developers' preferences in joining new projects. When developers' preferences are not aligned with project goals, developers are likely to leave the project or remain inactive even after joining. For example, it has been shown that selecting a project only based on project popularity or high rate of activity, e.g., number of commits, may lead to developer attrition (Bayati and Peiris 2018)because sustained contributions requires alignment between developer preferences and project attributes and requirements. Indeed, studies have documented high attrition rates in OSS community in recent years (Steinmacher et al. 2018).

One can argue that finding and selecting a project, out of millions available, is a complex and a time consuming process. Thus, developers rely on their social network to find and join OSS projects. Recent studies have shown two prominent trends among developers joining OSS projects- 1) based on prior ties and 2) based on popularity of developers/projects. First one argues that prior collaborations and ties with project members or project owners may affect developers' decision to join a project (Hahn et al. 2008). However, there are three problems with this approach- 1) it limits the opportunities for both developers and OSS projects, 2) developers' preferences change since their prior ties and 3) newcomers are at a disadvantage with this approach. For example, GitHub lists almost 100 million OSS projects; therefore, relying only on previous ties to find projects is not only difficult for developers but also limiting and inefficient for the OSS ecosystem. Second, a set of studies have shown that developers follow popular developers (Blincoe et al. 2016; Nielek et al. 2016) and projects (Fronchetti et al. 2019; Jarczyk et al. 2014) to find and join OSS projects. However, there are three problems with this approach, 1) popularity can be driven by many factors, 2) needs of popular projects and developers' skills may not be aligned and finally, 3) this approach is not scalable when it comes to matching millions of developers to projects. Further, this may lead to distortions in OSS ecosystem where some projects get flooded with developers while others close due to lack of volunteer contributions. Our work contributes to this literature by arguing that developers' preferences should play an important role in project selection and therefore we propose an approach to recommend projects to developers based on their preferences.

Attracting and retaining OSS developers

It is not only developers who face challenges in finding OSS projects that match with their preferences, OSS project owners also strive for motivated developers whose skills and expertise match with the project requirements (Figure 1), because absence of that leads to developer attrition. Unfortunately, developers turnover is high in OSS projects and therefore, retention of developers is one of the biggest challenges for open source projects (Schilling 2014). In fact, four out of five projects fail due to lack of sustained contributions (Schilling et al. 2012b).

According to the Recruitment theory the person-job fit and person-team fit are critical for sustained contributions (Edwards, 1991). This matching or fitness can be assessed both from developer and project perspective, using objective or subjective measurements. For example, using crosssectional data, Sharma et al. (2010) examined the effect of developers' perceived fit with OSS projects on developers' turnover. While their study was among the first to highlight the need for a fit between developers and projects, it cannot be scaled for matching devlopers and projects as we propose in this study because- a) their study relied on subjective assessment, which is questionable, b) it only considered developer's perspective, without considering projects, and finally, c) in a dynamic environment where both project and developers evolve over time, a cross-sectional assessment has many limitations. Some of these issues were addressed by Schilling et al. (2012b), who focused on students contributing to Google Summer of Code project. Authors used students' historical contributions and associations with the project to objectively measure person-job fit and person-team fit and concluded the importance of these fitness meaures on sustained contributions in OSS projects. While they highlighted the importance of using historical activities data to measure fit, their study was focussed on one project and didn't account for the evolving nature of developers and projects in OSS ecosystem. Further, their study was narrowly focussed on one technical activity whereas literature has argued that both social and technical activities are important for matching developers to projects.

Software development in general and OSS in particular are social coding activities, which require developers' social along with technical skills for sustained contributions (Bird et al. 2009). Developers' social commitments (Schilling et al. 2013), social ties (Schilling 2012) and social identity with OSS community drives their sustained participation in OSS projects (Chou and He 2011; Fang and Neufeld 2009). Therefore developers' social and technical skills should be aligned with that of project members and project goals (Schilling et al. 2012b), which is often not the case and lacks in developers' project selection process leading to higher attrition rates. Our research aims to address this issue by recommending projects to developers d based on developers' social and technical activities/preferences.

Scholars used Herzberg's two factor theory to examine developers' dissatisfaction in open source community (Yu et al. 2012) and found that developers who use the software for personal use have lower dissatisfaction than others. The significant role of personal needs highlights the value of personalized rating technique for project selection. Other similar studies find that project characteristics and developers skills affect turnover rate in OSS community (Sharma et al. 2012). Overall, we observe that on one side, the literature has shown the effect of project attributes on turnover (Chengalur-Smith et al. 2010; Oh et al. 2016; Schilling et al. 2012a; Shah 2006; Yamashita et al. 2016), but on the other side, another stream of literature shows the effect of developers' skills/preferences on turnover (Sharma et al. 2012; Steinmacher et al. 2015; Yu et al. 2012). Drawing from these two streams, we argue that lower turnover in OSS may depend on adequately matching socio-technical attributes of both developer and projects. Therefore, we use socio-technical activities for both projects and developers over time to create personalized project rankings.

Recommender Systems in Software Engineering (RSSE)

Literature on Recommender systems in software engineering (RSSE) (Robillard et al. 2014) has mainly focused on software development related issues, such as, Bug triaging, finding experts (Anvik et al. 2006; Anvik and Murphy 2011; Badashian et al. 2015; Naguib et al. 2013; Shokripour et al. 2013), source code suggestion (Holmes and Murphy 2005), design pattern recommendation (Palma et al. 2012), tracking updates (Zimmermann et al. 2005), requirement management (Maalej and Thurimella 2009), recommending pull request reviewers (Yu et al. 2016), following (Schall 2014) and experts (Allaho and Lee 2014). While these studies tackle interesting issues, they lack on following two challenges- 1) these studies do not address the demand and supply issues in OSS, which is recommending projects to developers. Note that recommending projects to developers is challenging due to continuous changes in Developers' preferences and projects' goals over time. And, 2) most of these studies have used MSR (mining software repositories) techniques, such as source code analysis, developers' communication, and project historical data analysis to develop recommender systems (Anvik et al. 2006; Yu et al. 2016). Instead of using traditional MSR techniques, we have used large-scale data analysis techniques to query GitHub to obtain developers' preferences from their social and technical activities and developed personalized implicit project ratings for individual developers.

OVERALL FRAMEWORK

While it is common knowledge that developers' technical skills are critical for development and success of OSS, research is thin on the role of developers' social skills in OSS development. Researchers argue that software applications are an outcome of collaboration among developers and other project members. Specifically in OSS projects where a software artefact is developed with technical knowledge and interactions with a large and diverse project community, both social and technical activities are important and influential in project success or failure (Bird et al. 2009). Knowledge creation in OSS is influenced by social structure and member's network cohesion (Singh et al. 2011), and social structure of the OSS project and associated community has an effect on team activity (Wu and Goh 2009). Therefore, developers' social skills are critical for their acceptance and success in the OSS community (Carillo et al. 2017; Qureshi and Fang 2011). Holistically, communication, collaboration and coordination between various stakeholders (Syeed and Hammouda, 2013), including community members, is crucial for success and survival of these projects.

Theory of translucence (Erickson and Kellogg 2000) lays the foundation of a system that facilitates asynchronous communication, coordination and collaboration between geographically diverse, large groups of developers, required for OSS development. We argue that OSS social coding platforms, e.g., GitHub, are social translucent systems. These coding platforms have key characteristics- *visibility*, *awareness* and *accountability* (Erickson and Kellogg 2000) outlined in the theory of translucence, critical for communications and collaborations among OSS developers. For example, GitHub allows developers to be *visible* to others, *aware* of other developers and *accountable* for their actions and behaviours, thereby supports consistent behaviour in groups and communities (Barreto, Karapanos and Nunes, 2011), and enables interaction and collaboration among OSS community members. Developers' activities on GitHub are open and transparent allowing community members to learn and evaluate their skills, goals and preferences. This openness has spurred collaboration and contribution in OSS projects.

Extant literature has used developers' social and technical activities to infer their goals, expertise and fit with OSS projects (Dabbish et al. 2012; Tsay et al. 2014b) and also for evaluating their contribution on OSS projects (Tsay et al. 2014a; Tsay et al. 2014b). While scholars agree that OSS developers' social and technical skills are critical for their contribution in OSS ecosystem, literature is scattered on what constitutes social and technical activities. We extend this literature and propose to classify developers' activities as *social*, *technical* and *collaborative*. We contribute[1] to this stream of literature

[1] We test our design artefact against both classifications- 1) *Social and Technical*, as suggested in the extant literature, and 2) Social, Technical and Collaborative, that we propose in this study. See details in section – Robustness Checks.

by adding collaborative activities along with social and technical activities. These activities, over time across different projects, are used to measure developers' preferences and developing a personalized implicit project ranking for OSS developers. These rankings are then used as an input to a recommender system to recommend projects to developers. Next, we discuss developers' *social, technical and collaborative activities* in OSS projects.

Social Activities

In OSS ecosystem, communities are built around OSS projects and play an important role in success of these projects (Carillo et al. 2016; Daniel et al. 2018). Developers' affiliations with peers in project community affect project success (Grewal et al. 2006) and social structure, which eventually affects their contribution and knowledge creation in OSS projects (Singh et al. 2011).

Developers' social activities in OSS projects indicate their desire and ability to connect and affiliate with other developers, and integrate with the project team (Gharehyazie, et al., 2015). Understanding the value of social activities, social coding platforms such as GitHub use a '*following*' feature (Blincoe, et al., 2016), similar to the one on social media platforms such as *Twitter*. *Following* activity is used as a measure of developers' *social activity*(Lee et al. 2013; Moqri et al. 2018). *Following* is different from participation, and it is referred as affiliation (Goggins and Petakovic 2014) and awareness(Wu et al. 2014) , and therefore, supports the *awareness* characteristic of the theory of translucence.

By *following* someone, a developer receives the latest information, such as participation and contribution activities about the *followed* developers. *Following* influences actions and contributions, leads followers to new projects (Badashian and Stroulia 2016; Blincoe et al. 2016), and is often considered more influential than contribution in OSS ecosystem (Blincoe, et al., 2016). Indeed, social recommender systems are using *following* feature, to measure friendship and trust to find similar users (Montaner et al. 2002; Schall 2014; Yang et al. 2014).

Following also leads to social influence (Cha et al. 2010) and social ties which affect developers' level of interest in a project (Hahn et al. 2006; Hannebauer and Gruhn 2017). We argue that *following* and *followed by* activities among peers within a project affect social structure and cohesion and therefore, developers' affiliation and commitment (Singh et al. 2011) and contribution (Wu and Goh 2009) for the project. We use these activities as a measure of developers' *social activities* in a project (Figure 6).

Technical Activities

A social coding platform such as GitHub hosts millions of OSS projects and facilitates features such as distributed development, issue tracking, online build process, code review and source control, etc. just to name a few. Developers' *technical activities* are related to any type of coding and development submissions (Sarker et al. 2019; Vasilescu et al. 2016) and there is a little ambiguity about core technical activities in OSS communities (Badashian et al. 2014; Tsay et al. 2014a), which are *Commits*, *Issues* and *Pull Requests* (see Figure 6). Following the literature, we count these core technical activities (Vasilescu et al. 2015) to measure developers' technical activities (Figure 6).

Figure 3 shows two of these technical activities to illustrate the process of commits, pull requests and associated discussions. It shows that a developer has forked a project from main (master) branch, committed changes in the forked version and then submitted a pull request to the core members to review her changes/commit. This pull request triggers discussion before commits in the forked project are merged with the main project (master branch). Next, we describe all the three technical activities- commit, issue and pull request (Vasilescu et al. 2015).

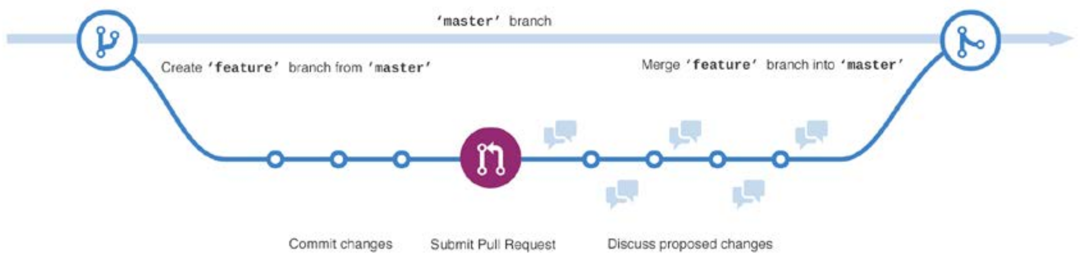

'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

Figure 3. Commit, Pull Requests and Discussion in an OSS project on GitHub

On GitHub, developers can make a copy of an OSS project to make changes to the source code. These changes are saved on their copy, also termed as forked project. This event (saving changes) is named as ***Commit***. Commits can vary in their size and importance. Following OSS literature (Badashian et al. 2014; Vasilescu et al. 2013), we use the number commits to measure a developer's technical activity and her interest in a project.

Open source community allows contributions in variety of ways (Crowston and Howison 2005). For example, some of the volunteers play (test) with the OSS software and list bugs and required features via a system called *issue tracking*. On GitHub, the term ***Issue*** is used for bugs and features reported by developers or other contributors. Project coordinators assign *Issues* to developers based on their expertise (Anvik and Murphy 2011). Successful resolution of issues are critical for project quality and success (Bissyandé et al. 2013; Jurado and Rodriguez 2015). Number of *Issues* assigned to a developer signals her expertise and measures her interest in a project and therefore, can be used to measure developer's technical activity in a project.

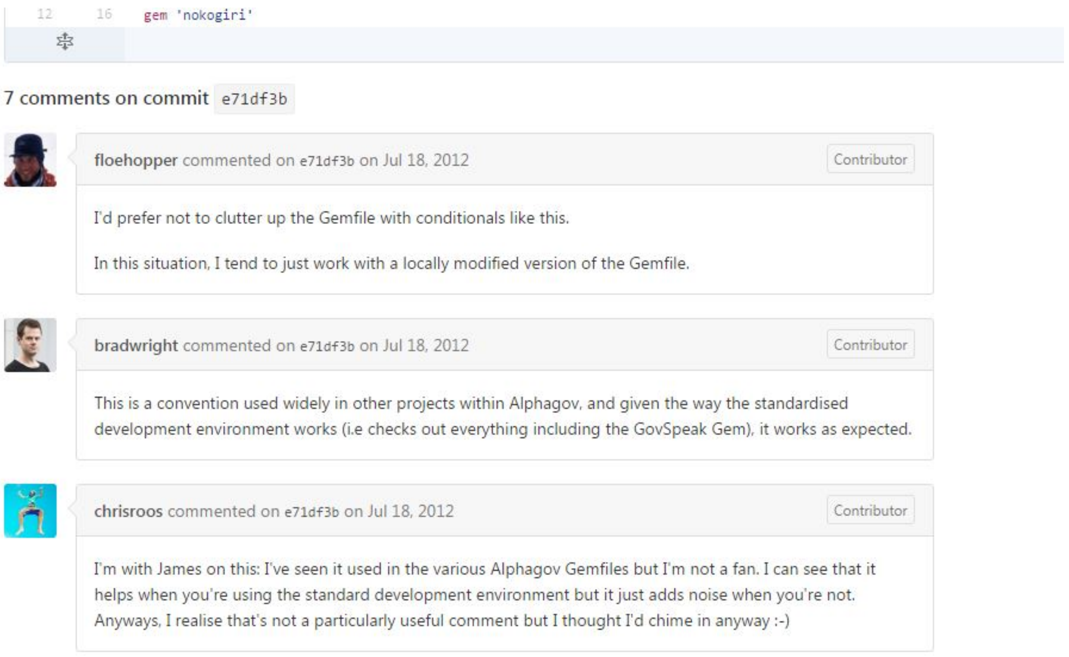
In distributed software engineering, developers work on their assigned *issues* and development tasks on their local repositories (forked version). Whenever they want to merge their code into the main repository, they send a request for merging, which is called ***Pull Request***. A pull request is checked for suitability following standard policies. After acceptance, it is merged with the master repository for production purposes. Software engineering literature has named this strategy as pull-based software development (Yu et al. 2016). *Pull Request* has been used in software engineering literature to examine developers' contribution (Badashian et al. 2014; Gousios et al. 2016). We use the number of pull requests to measure a developer's technical activity and her interest in a project.

We understand that the number of commits, pull requests and reported issues is not the perfect way to measure technical activities/contributions, because these contributions may vary in terms of their size, complexity, efficiency and their overall effect. For example, a code commit may vary from just adding or removing a few lines of code to huge in-depth changes in multiple correlated source code.

However, most of the studies in literature have used these proxies to measure developers' contributions (Daniel and Stewart 2016; Grewal et al. 2006; Singh et al. 2011) due to measurement challenges associated with other matrices.

Collaborative Activities

Interactions and discussion among developers are critical in distributed software development environment (Guzman et al. 2014; Guzzi et al. 2013). In OSS communities, most of these interactions are text-based, in the form of reviews, comments, messages, and e-mails. These conversations help enhance products' technical features (Guzman et al. 2014; Guzzi et al. 2013). Figure 4, shows an example of how contributors discuss a submitted commit via *commit comments*.



```
12    16    gem 'nokogiri'
```

7 comments on commit e71df3b

floehopper commented on e71df3b on Jul 18, 2012 — Contributor

I'd prefer not to clutter up the Gemfile with conditionals like this.

In this situation, I tend to just work with a locally modified version of the Gemfile.

bradwright commented on e71df3b on Jul 18, 2012 — Contributor

This is a convention used widely in other projects within Alphagov, and given the way the standardised development environment works (i.e checks out everything including the GovSpeak Gem), it works as expected.

chrisroos commented on e71df3b on Jul 18, 2012 — Contributor

I'm with James on this: I've seen it used in the various Alphagov Gemfiles but I'm not a fan. I can see that it helps when you're using the standard development environment but it just adds noise when you're not. Anyways, I realise that's not a particularly useful comment but I thought I'd chime in anyway :-)

Figure 4. Evidence of commit comments

Literature generally considers the discussion/comment in OSS as a social activity (Gharehyazie et al. 2015; Sarker et al. 2019; Vasilescu et al. 2015). For example, communications regarding issues, etc. (Gharehyazie, et al., 2015) and comments on commits, pull requests, etc. (Vasilescu, et al., 2015) have been considered as social activities. However, we contend that discussion/comment on an issue or pull request is not a mere social activity because issues or pull requests are assigned to developers based on their technical expertise (Anvik and Murphy 2011) and are not open to anyone intending to just socialize. Developers require technical expertise to discuss about different facets of a bug or a requested feature while they are reviewing the source code. Along with the technical knowledge, they also need to follow the social norms of the community during these discussions and therefore, these activities are somewhere in between purely technical or social and truly measure the collaborative aspect of OSS ecosystem. These activities require collaboration in a form of discussion/commenting, where more than one developer work on the same artefact, using different ICT tools, such as emails, instant messaging, and web-based applications (Xuan and Filkov, 2014) and therefore, we term them as *collaborative activities*. We use three types of interactions-*commit comments*, *issue comments* and *pull request comments*, to measure developers' collaborative activities in a project (Figure 6).

Several developers may comment on a commit to increase the quality of the commit, and of the project as whole. Though only registered contributors can write commit comments, but these comments are public for scrutiny and participation purpose. Commit comments reveal developers' expertise, behaviour, participation and engagement with the development process (Guzman et al. 2014; Pletea et al. 2014).

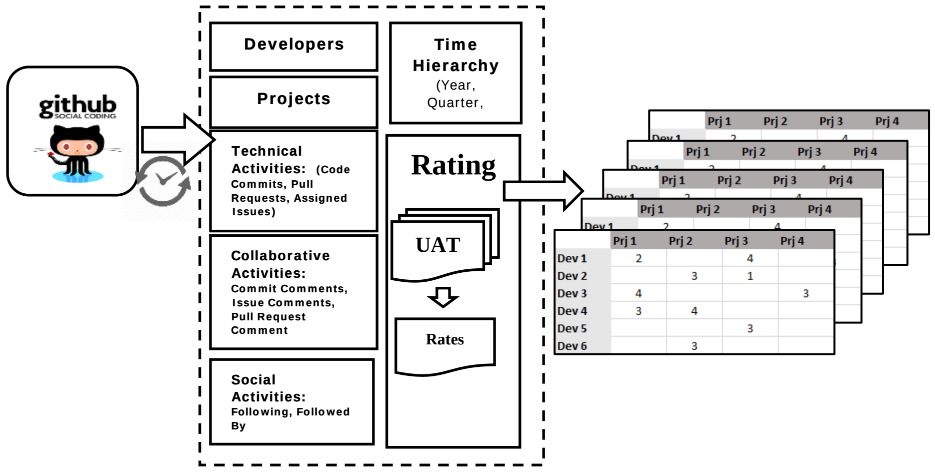
Similar to commit, developers collaborate by commenting to increase the quality of *issue* resolution, at any stage, from opening to closing. *Issue Comments* are technical in nature and help developers collaborate in their contribution to OSS project.

Before accepting or rejecting a pull request, project coordinators and senior project members discuss different aspects of the request via *Pull Request Comments*. Each *pull-request* may invoke a bunch of reviews/comments in form of back and forth exchange between developers before acceptance. Pull request comments provide a critical tool for evaluating, assessing and reviewing (Yu et al. 2016) pull requests.

While we measure the contributions by the count of comments, etc., we understand that comments' size and importance may also vary. For example, a comment from an expert or a core member may have different effect compared to a comment from a new or inexperienced developer. However, count is used as a proxy in the literature in OSS (Sarker et al. 2019; Vasilescu et al. 2015) and social media (De Vries et al. 2012; Swani et al. 2017).

Developers' socio-technical activities in OSS community have been used for project reputation analysis and its usability evaluation (Dabbish et al. 2012). We use these socio-technical activities to measure developers' preferences for projects. These preferences are used to rank order their preferred projects, which is very important for developing a recommendation system for OSS projects (Hahn et al. 2008).

We follow design science research (DSR) 6-step process model (Peffers et al. 2007) as shown in Figure 2. Our main contribution is an artefact, which is a time-based implicit feedback rating mechanism for open source projects based on developers' social, technical and collaborative activities in a social coding environment.


github SOCIAL CODING
Developers
Projects
Technical Activities: (Code Commits, Pull Requests, Assigned Issues)
Collaborative Activities: Commit Comments, Issue Comments, Pull Request Comment
Social Activities: Following, Followed By
Time Hierarchy (Year, Quarter,
Rating
UAT
Rates

	Prj 1	Prj 2	Prj 3	Prj 4
Dev 1	2		4	
Dev 2		3	1	
Dev 3	4			3
Dev 4	3	4		
Dev 5			3	
Dev 6		3		

Figure 5. Implicit Project Rating Module

Our proposed framework is shown in Figure 5. We divide developers' activities in three main categories- *Social*, *Technical* and *Collaborative* (Figure 6). Using time discounted activities (Dabbish et al. 2012), our rating module calculates developers' preferences in projects on a 1 to 5 scale. This rating module is one of the main contributions of this study that is presented as an algorithm, which we will discuss in next section.

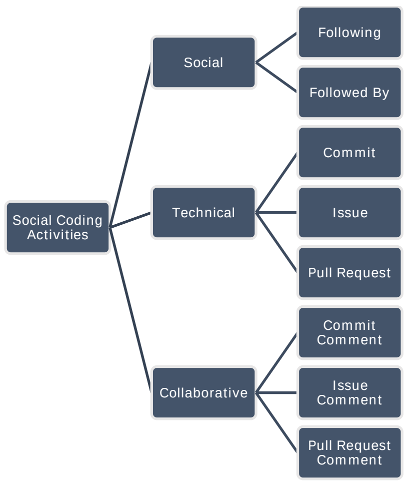


Figure 6. Hierarchical categories of social coding activities.

In this study, we have applied implicit feedback approach for collaborative filtering (CF) recommendation (Kelly and Teevan 2003; Lee et al. 2008; Palanivel and Sivakumar 2010). Implicit feedback rating approach is used when, 1) direct ratings are not possible/available, and/or 2) the rating values are temporal and dynamic (Hu et al. 2008; Lee et al. 2008). Both of these issues are relevant in our context. First, GitHub does not provide a mechanism for developers to rate OSS projects and second, developers' activities on projects change over time, which may affect their project rankings. Our approach deals with both of these challenges. We discuss the details of rating mechanisms in the next section.

We use personalized project ratings generated via implicit feedback approach for collaborative filtering (CF) recommendation because CF has been used to predict users' interests based on choices of other similar users (Ricci et al. 2011). CF is independent of the content of items and there is no need for complex similarity calculation because it is based on similarity in preference (project ratings). We now present a general memory-based CF approach which uses Pearson or Cosine correlation similarity (Ricci et al. 2011). Equation 1 shows our proposed approach to predict developer *d's* rating for a project *p*, where. *D* is the list of top *N* most similar developers (in terms of their project ratings) and *k* is the normalization factor shown in equation 2. *Sim(d,d')* calculates the similarity between developer *d* and *d'* (Equation 3a).

$$PredictedRate(d,p) = k \sum_{d' \in D} Sim(d,d') \times Rate(d',p) \tag{1}$$

$$k = \frac{1}{\sum_{d' \in D} Sim(d,d')} \tag{2}$$

The Pearson Correlation similarity is calculated based on Equation 3a, where *r* represents rating and P_{xy} represent a list of common projects between user *x* and *y*. One could use other approaches such as Cosine similarity (Equation 3b), widely used in in the literature (Ricci et al. 2011), or other similarity metrics like Jaccard and Euclidean distance (Ricci et al. 2011). We do not discuss other CF approaches because comparing different approaches is not the focus of this study.

$$Sim(x,y) = \frac{\sum_{p \in P_{xy}} (r_{x,p} - \overline{r_x})(r_{y,p} - \overline{r_y})}{\sqrt{\sum_{p \in P_{xy}} (r_{x,p} - \overline{r_x})^2 \sum_{p \in P_{xy}} (r_{y,p} - \overline{r_y})^2}} \tag{3a}$$

$$Sim(x,y) = \frac{\sum_{p \in P_{xy}} r_{x,p}\, r_{y,p}}{\sqrt{\sum_{p \in P_x} r_{x,p}^2 \sum_{p \in P_y} r_{y,p}^2}} \tag{3b}$$

ARTEFACT DESCRIPTION

This study proposes a novel implicit rating mechanism for rating open source projects based on developers' social, technical and collaborative activities. Project ratings are an input to a collaborative filtering recommender system to recommend the most appropriate project for a developer. Details of the rating technique are presented below in Algorithm 1. For ease of exposition, we scaled the developers' project ratings from 1 to 5, where 5 represents the highest interest value.

Algorithm 1: Time-Based Implicit Feedback for OSS Projects

```
Inputs:
        Developers (List of Developers N)
        Prjs (List of Projects M)
        Activities (List of Developers' Activities in GitHub Projects)
        ActivityTypes (K)
Output:
        CubeRatings (Cube $_{K*N*M}$ of Developers' implicit rates for Projects in each Activity)
Procedure:
For each activity$_k$ in ActivityTypes
        For each developer$_n$ in Developers
                For each project$_m$in Prjs
                        $X_{n,m}$= SUM(DPT(developer$_n$, project$_m$, time$_t$))
```

```
For each developer$_n$ in Developers
    For each project$_m$ in Prjs
        ActivityRatings$_{n,m}$=Rates(developer$_n$, project$_m$, X)
CubeRatings$_{k,n,m=}$ ActivityRatings$_{n,m}$
```

In Algorithm 1, two functions are used- *DPT(x,y,z)* and *Rates(x,y,z)*. For a given activity type, DPT (Developer, Project, Time period) calculates total time-discounted activity of a developer *d*, on a specific project *p*, in time period *t* (Equation 4). Algorithm 1 receives four inputs- a list of developers, a list of OSS projects, a list of activities performed by developers on projects with their timestamps, and a list of different activity types. The output of the Algorithm 1 is a cube of implicit feedback ratings (CubeRatings), which can be used for predicting developers' preference. The dimensions of the cube are activity type, developers and projects. Each cell in the rating cube contains the calculated rate value. In Algorithm 1, The function *SUM(argument)* summarizes all the *DPT* returned values for different time periods in one cell of matrix *X*, which we discuss later.

$$DPT(d,p,t) = \frac{\sum Activities\ of\ d\ on\ p\ in\ t}{f(t)+1} \tag{4}$$

$$f(t) = (CurrentPeriod - t) \tag{5}$$

$$f(t) = (CurrentPeriod - t)^{(CurrentPeriod - t)} \tag{6}$$

Equation 4, demonstrates DPT functionality in detail. Activities are time discounted, giving more weight to the ones in recent periods (Lee et al. 2008; Robbes and Röthlisberger 2013), as shown in Equation 4. This time-discounting approach accounts for changes in preferences and goals for both developers and projects. Therefore, OSS project recommendation system in our artefact is a multi-criterion decision-making process (MCDM),

For the ease of exposition, we have considered linear discounting (Equation 5) however, our artefact can handle any other forms of discounting (e.g., exponential, Equation 6). Further, linear approach has been commonly used in the literature (Robbes and Röthlisberger 2013) because it performs well. The Rates function (Equation 7) calculates developers' rates (ratings) to their contributed projects based on *DPT*. The summation of all *DPT* for an activity type for each developer on a specific project is stored in a matrix *X*, where rows and columns are developers and projects. This matrix is an input to Rates function (Equation 7).

$$Rates(d,p,X) = \frac{\frac{X[d,p]}{\frac{\sum_{i=0}^{N} X[Developers[i],p]}{N}}}{\sum_{k=0}^{M} X[\,d, Prjs[k]] \times \frac{1}{\frac{\sum_{j=0}^{N} X[Developers[j], Prjs[k]]}{N}}} \qquad (7)$$

In equation 7, *d* and *p* represent the developer and project, *M* is the number of projects and *N* is the number of developers. *X[a,b]* refers to the element $X_{a,b}$ of matrix *X*. Our approach for rate calculation considers two different ways to normalize the rates value. First, we divide a developer's total activity in a specific project by the average activities of other developers in the same project, during the same time. This allows us to capture developer's relative interest in that project compared to other contributors. Second, by normalizing developer's relative interest for a project with other projects this developer contributes, we capture developer's relative preference for this project compared to other projects. The value of the rating is stored in a matrix, ActivityRatings (see Algorithm 1), which is for a specific activity, say commit. Similar exercise is performed for each of the eight activities (2 for social, 3 for technical and 3 for collaborative) to generate ActivityRatings for each activity. Together all these ActivityRatings are compiled in CubeRatings. Note that CubeRatings is a three-dimensional matrix whereas ActivityRatings is a two dimensional matrix (see Algorithm 1). We illustrate the working of algorithm 1 via an example, later in this section.

The CubeRatings matrix is three dimensional but we desire a two-dimensional rating matrix with dimensions as developers and projects. Therefore, we calculate an OverallRatings matrix, which is two dimensional. We argue that developers' preferences for activities may vary and it is less likely that developers have equal preference for the all the activity types. For example, a developer may partake in social, technical and collaborative activities but they contribute mostly via commits (a technical activity), and thus commit activity should get a higher weightage when computing project ratings for this developer. Therefore, we compute weights for each activity type for every developer to reflect their activity preference in their project ratings. Equation 8 demonstrates a generic approach to estimate weights for each activity. Here d, a and p, represent developers, selected activity, and project, respectively. $W_{d,a}$ demonstrates the weightage for activity *a* for developer *d.*

$$W_{d,a} = \frac{\sum_p CubeRatings[a,d,p]}{\frac{1}{n}\sum_d^n \sum_p CubeRatings[a,d,p]} \tag{8}$$

Algorithm 2, is an implementation based on equation 8 to calculate ModeratedWeights matrix (see Algorithm 2). This matrix is two dimensional. In Algorithm 2, we calculate the average number of activities for all developers and check the variance of the current user with the average.

Algorithm 2: Developers' Interest Calculation on Each Activity Type

```
Inputs:
    Developers (List of Developers with size N)
    Prjs (List of Projects with size M)
    Activities (List of Developers' Activities in GitHub Projects)
    ActivityTypes (with size K)
Output:
    ModeratedWeight (Matrix $_{N*K}$ of Developers' Weight for Each Activity Type)
Procedure:
For each activity$_a$ in ActivityTypes
    C=0, S=0
    For each developer$_d$ in Developers
        $A_{d,a}$=0
        For each project$_p$ in Prjs
            $X_{d,p}$= SUM(DPT(developer$_d$, project$_p$,time$_t$))
            $A_{d,a}=A_{d,a}+X_{d,p}$
        S=S+ $A_{d,a}$
        C=C+1
    Avg$_a$=S/C
For a=1 to K
    For d=1 to N
        Weight$_{d,a}$=$A_{d,a}$/Avg$_a$
For a=1 to K
    SumWeight=0
    For d=1 to N
        SumWeight$_d$= SumWeight$_d$+ Weight$_{d,a}$
For a=1 to K
    For d=1 to N
        ModeratedWeight$_{d,a}$= Weight$_{d,a}$/ SumWeight$_d$
```

The moderated weight is calculated to understand the importance of each activity for a developer. Overall rating is derived from multiplying the moderated weight with the value of cube rating cells as shown in Algorithm 3. We now elaborate Algorithm 3 with an example. For example, consider <0.6, 0.1, 0.3> as the moderated weights for a developer for three different type of activities. Further, this developer contributed in 2 projects and rates in her CubeRatings matrix for both the projects (based on all three activities) are <<2,3,3>, <5,1,2>>. By applying algorithm 3, we get <<0.6*2+0.1*3+0.3*3>, <0.6*5+0.1*1+0.3*2>> which results into <2.4, 3.7> as overall rating of this developer for these two projects. However, if we consider equal weights (for each activity) we get <(2+3+3)/3, (5+1+2)/3)>, or <2.67, 2.67>for both the projects. We see that with equal weights, both projects are ranked at the same level (<2.67, 2.67>) whereas by considering developers' preferences for different activities (Algorithm 3), second project is ranked higher (<2.4, 3.7>). Overall ratings can be calculated using equation 9. Where the *CubeRatings* function represents the calculated rate on activity *a* in project *p* for developer d (refer to formula 4). $W_{d,a}$ demonstrates the weightage for activity *a* for developer *d*, as shown in equation 8. We have proposed a generic form, but our approach is modular and can adopt to any different approach for estimating weights, such as equal weighting for each activity. The *OR* function populates the cells of OverallRating matrix.

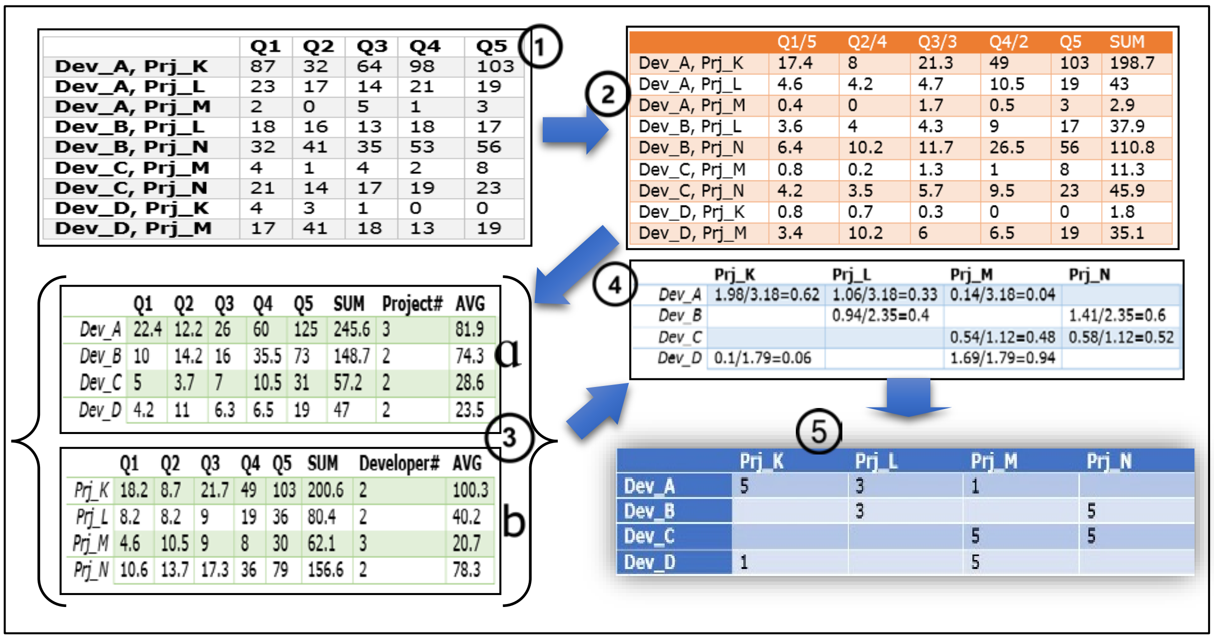
$$OR(d,p) = \sum_a W_{d,a} \times CubeRatings(a,d,p) \tag{9}$$

Algorithm 3 shows the calculation of Moderated Rates while different criteria for developers' interests applied on their activities.

```
Algorithm 3: Multi-Criteria Overall Rating Calculation
Inputs:
        Developers (List of Developers with size N)
        Prjs (List of Projects with size M)
        Activities (List of Developers' Activities in GitHub Projects)
        ActivityTypes (with size K)
        ModeratedWeight (Matrix $_{N*K}$ of Developers' Weight for Each Activity Type)
        CubeRatings (Cube $_{N*M*K}$ of Developers' implicit rates for Projects in each Activity)
Output:
        OverallRatings (Matrix $_{N*M}$ of Developers' overall rates for Projects)
```

```
Procedure:
For each activity_m in ActivityTypes
    For each developer_a in Developers
        OverallRatings_{a,b} = 0
        For each project_b in Prjs
            OverallRatings_{a,b} = (OverallRatings_{a,b} + (ModeratedWeight_{a,m} * CubeRatings_{a,b,m} ))
```

We now illustrate our Algorithm 1 via an example. Figure 7 shows the main steps of applying Algorithm 1. In follows, we show steps for generating an ActivityRatings matrix (see Algorithm 1) for one activity. As shown in Algorithm 1, there is one ActivityRatings matrix for each activity. CubeRatings matrix is just accumulation of all these ActivityRatings matrices (see Algorithm 1). CubeRatings matrix is then multiplied by moderated weights matrix to generated OverallRatings (see Algorithm 3).



	Q1	Q2	Q3	Q4	Q5
Dev_A, Prj_K	87	32	64	98	103
Dev_A, Prj_L	23	17	14	21	19
Dev_A, Prj_M	2	0	5	1	3
Dev_B, Prj_L	18	16	13	18	17
Dev_B, Prj_N	32	41	35	53	56
Dev_C, Prj_M	4	1	4	2	8
Dev_C, Prj_N	21	14	17	19	23
Dev_D, Prj_K	4	3	1	0	0
Dev_D, Prj_M	17	41	18	13	19

	Q1/5	Q2/4	Q3/3	Q4/2	Q5	SUM
Dev_A, Prj_K	17.4	8	21.3	49	103	198.7
Dev_A, Prj_L	4.6	4.2	4.7	10.5	19	43
Dev_A, Prj_M	0.4	0	1.7	0.5	3	2.9
Dev_B, Prj_L	3.6	4	4.3	9	17	37.9
Dev_B, Prj_N	6.4	10.2	11.7	26.5	56	110.8
Dev_C, Prj_M	0.8	0.2	1.3	1	8	11.3
Dev_C, Prj_N	4.2	3.5	5.7	9.5	23	45.9
Dev_D, Prj_K	0.8	0.7	0.3	0	0	1.8
Dev_D, Prj_M	3.4	10.2	6	6.5	19	35.1

	Q1	Q2	Q3	Q4	Q5	SUM	Project#	AVG
Dev_A	22.4	12.2	26	60	125	245.6	3	81.9
Dev_B	10	14.2	16	35.5	73	148.7	2	74.3
Dev_C	5	3.7	7	10.5	31	57.2	2	28.6
Dev_D	4.2	11	6.3	6.5	19	47	2	23.5

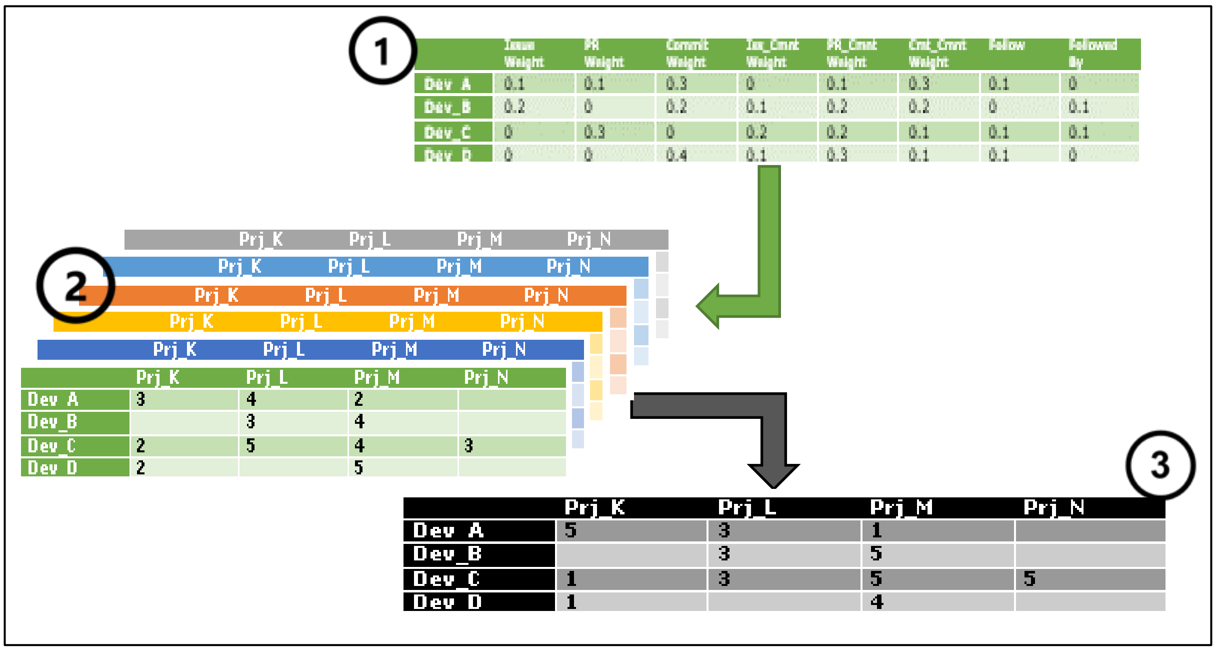
	Q1	Q2	Q3	Q4	Q5	SUM	Developer#	AVG
Prj_K	18.2	8.7	21.7	49	103	200.6	2	100.3
Prj_L	8.2	8.2	9	19	36	80.4	2	40.2
Prj_M	4.6	10.5	9	8	30	62.1	3	20.7
Prj_N	10.6	13.7	17.3	36	79	156.6	2	78.3

	Prj_K	Prj_L	Prj_M	Prj_N
Dev_A	1.98/3.18=0.62	1.06/3.18=0.33	0.14/3.18=0.04	
Dev_B		0.94/2.35=0.4		1.41/2.35=0.6
Dev_C			0.54/1.12=0.48	0.58/1.12=0.52
Dev_D	0.1/1.79=0.06		1.69/1.79=0.94	

	Prj_K	Prj_L	Prj_M	Prj_N
Dev_A	5	3	1	
Dev_B		3		5
Dev_C			5	5
Dev_D	1		5	

Figure 7. An Illustrative Example of Algorithm 1

We start by collecting data on developers' specific type of activity, e.g., commit on projects, from previous five quarters (Table 1, Figure 7). Let's say that we have collected data for five quarters (time periods), from Q1 (t=1) to Q5 (t=5). Our goal is to compute implicit project ratings at the end of quarter 5 (Q5). In Table 2 of Figure 7, linearly discounted activities are calculated using equations 4 and 5. For example, in Table 2, the discounted activity for Dev_A, Prj_K, in Q1 is 17.4 (87/(5-1+1)) and the discounted activity for Dev_A, Prj_K, in Q2 is 8 (32/(5-2+1)). In the next stage, we group all the discounted activities for developers (Table 3a) and projects (Table 3b) separately to capture their trend. For example, in Q1, Dev_A has done 22.4 (17.4+4.6+0.4) discounted activities across three projects, K, L and M (Table 3a). Similarly, there were 18.2 (17.4+0.8) discounted activities in Proj_K from all the developers (developers A and D) in Q1 (Table 3b). We also calculate the average contribution (discounted activity) from developer and project perspective. For example, developer A worked on 3 projects and her average contribution on a project was 81.9 (last column of Table 3a). We performed similar calculations in Table 3b for projects. In fourth step (Figure 7, Table 4), we aim to estimate developers' relative performance on each project. We first divide the total activity of a developer on a project by the average of activities on that project. For example, we divide developer A's total activity on project K (198.7, first row, last column of Table 2) by average contribution on project K by any member (100.3, first row, last column of Table 3b), which results into 1.98 (198.7/100.3), and is shown in the first row and column of Table 4. This means, that developer A's contribution on project K is almost double of the average contribution on project K during this period (Q1-Q5). We now do similar calculation for all the projects for developer A. We sum up all the relative activity/project for all the developers. For Dev_A, it results into 3.18 (1.98+1.06+0.14) as shown in Table 4 of Figure 7. We now divide developer A's relative preference for all projects by this total (3.18 for developer A) as shown in Table 4 of Figure 7. The highest value is the most interesting project for a developer, is rated as 5 and rest are scaled accordingly. Since we used commit activities for this example, the outcome of this exercise is a ActivityRatings matrix (Table 5, Figure 7) for commit activity. We generate similar ActivityRatings for all eight activities (2 for social, 3 for technical and 3 for collaborative). All these eight ActivityRatings put together form the CubeRatings matrix (see Algorithm 1). CubeRatings is a three-dimensional matrix which is multiplied with ModeratedWeight matrix (Algorithm 2 and 3) to generate the OverallRatings matrix, which is two dimensional with developers and projects as two dimensions. The OverallRatings matrix is an input to a CF recommender system.



	Issue Weight	PR Weight	Commit Weight	Iss_Cmnt Weight	PR_Cmnt Weight	Cmt_Cmnt Weight	Follow	Followed By
Dev_A	0.1	0.1	0.3	0	0.1	0.3	0.1	0
Dev_B	0.2	0	0.2	0.1	0.2	0.2	0	0.1
Dev_C	0	0.3	0	0.2	0.2	0.1	0.1	0.1
Dev_D	0	0	0.4	0.1	0.3	0.1	0.1	0

	Prj_K	Prj_L	Prj_M	Prj_N
Dev_A	3	4	2	
Dev_B		3	4	
Dev_C	2	5	4	3
Dev_D	2		5	

	Prj_K	Prj_L	Prj_M	Prj_N
Dev_A	5	3	1	
Dev_B		3	5	
Dev_C	1	3	5	5
Dev_D	1		4	

Figure 8. Illustration of Algorithm 3

Figure 8 schematically explains how ModeratedWeight matrix (Algorithm 2) is applied to CubeRatings to generate the OveralRatings. Table 1 in Figure 8, shows developers' perceived interest in each activity (ModeratedWeight matrix). These weights are applied to the CubeRatings matrix (see Algorithm 3) to generate the final OverallRatings matrix (Table 3, Figure 8), which is the main contribution of this study. We design an artefact for implicitly rating projects based on developers' social, technical and collaborative activities across different projects. These rating (OverallRatings) are used for recommending projects to OSS developers. We now discuss the data and evaluation of our proposed artefact.

DATA COLLECTION

We collected data from GitHub, which is the largest open source software repository. As of 2019 GitHub has more than 50 million users and 100 million of projects. Although GitHub provides a RESTful API for data collection, currently, two main archives store and collect GitHub's data in different formats. GhTorrent (Gousios 2013) and GitHub Archive (Grigorik 2012) provide different type of data access to GitHub. We selected GhTorrent (Gousios 2013) because it provides a regular

SQL-based data dump along with MongoDB data dumps facilitating Big Data analytics needed for this study and GhTorrent, it is the most used data set in GitHub analytics (Cosentino et al. 2017).

We collected data on projects that were created between 13/1/2008 to 18/6/2015, which resulted into a total of 20,048,071 projects and 7,210,741 unique users. We report summary statistics in Table 1, which shows the size and diversity of the activities in OSS projects hosted on GitHub and therefore confirms the challenges for developers to find a suitable project.

First Date	**Last Date**	**Number of User**	**Number of Project**	**Non-forked Projects (original)**	**Avg(Project/Usr)**	**Avg(Usr/Project)**
13/01/2008	18/06/2015	7,210,741	20,048,071	10,647,788	1.5072	3.2961
Commit	**Commit Comment**	**Issue**	**Issue Comment**	**Pull Request**	**Pull Request Comment**	**Follow**
260,619,933	2,017,971	18,895,174	32,304,278	7,946,946	4,312,419	7,022,950

Table 1. A Summary of GhTorrent Data Dump

From this data dump, we remove personal, dead, forked and unreal projects (Kalliamvakou et al. 2014) resulting into a dataset containing projects that are original, and are not forked from any other project (Kalliamvakou et al. 2016). We removed all the public projects converted to private or deleted from GitHub. To reduce the size of dataset for evaluating our proposed artefact, we only focused on developers who joined the GitHub in a specific month (January 2012) and collected all their social, technical and collaborative activities on GitHub for a period of three years. We selected year 2012 because GitHub got matured in year 2012 (Moqri et al. 2018; Nielek et al. 2016). Figure 9 shows the distribution of contributed projects by developers who joined in January 2012. This distribution follows a power law distribution. Figure 9 shows that more than 85% of contributors participated in 5 or fewer projects.

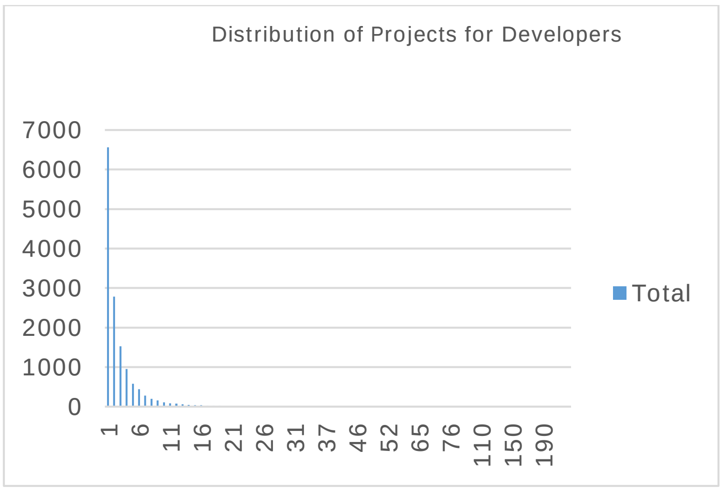


Figure 9. Distribution of number of contributed projects for GitHub members

Further analysis of this data showed that developers who work on 5 or more projects, contribute (in terms of commits) almost five times higher than those who work on 4 or less projects. In other words, the total number of commits by bottom 85% is about the same as the number of commits by top 15% developers. These statistics shows the influence of these *active developers* in the OSS community. Since developers' contributions are key to success and survival of OSS community, we want to focus on top 15% developers who do the heavy lifting in the community. Following this observation, we focus on these active developers to test our artefact. Note that these 15% are among the developers who joined GitHub in January 2012, a very small subset of the whole community. Our sample for evaluating the artefact contains 254 projects and 84 developers, which is the outcome of our data cleaning phase. Our sample size is comparable to other similar studies that have used data from GitHub (Blincoe, et al., 2016). We acknowledge that we have not considered developers' activities/contributions outside of GitHub, which may affect their overall preferences. However, GitHub is by far the largest repository and adequately considers developers' preferences, thus has been used in a large number of studies (Sarker et al. 2019; Vasilescu et al. 2016).

As discussed earlier, we only focussed on developers with at least five projects in their baskets, to reduce the sparsity of the rating matrix. Rating matrix sparsity and cold start are challenging issues in collaborative filtering (CF) systems (Lam et al. 2008; Lu et al. 2015). We put a limit on the minimum number projects a developer works on, for following reasons. First, since we want to evaluate our approach with state of the art techniques, we need to have a fairly populated project-developer matrix to make a meaningful comparison. Second, to employ collaborative filtering technique, we need to have some prior on developers' contribution to find similar/preferred projects. Third, developers participating in multiple projects are more active than others, and it is critical to recommend them suitable projects because they make the bulk of the contribution to OSS projects (Vasilescu et al. 2016).

	Commit	**Commit Comment**	**Issue**	**Issue Comment**	**Pull Request**	**Pull Request Comment**	**Following**	**Followed By**
Commit	1	0.1978	0.1651	0.3661	0.4607	0.2425	0.0747	0.0878
Commit Comment	0.1978	1	0.2658	0.3854	0.2671	0.2484	-0.0101	0.0526
Issue	0.1651	0.2658	1	0.2021	0.3002	0.1299	-0.0190	0.0522
Issue Comment	0.3661	0.3854	0.2021	1	0.5706	0.3100	0.0337	0.0991
Pull Request	0.4607	0.2671	0.3002	0.5706	1	0.4159	0.0472	0.1393
Pull Request Comment	0.2425	0.2484	0.1299	0.3100	0.4159	1	-0.0203	0.0854
Following	0.0747	-0.0101	-0.0190	0.0337	0.0472	-0.0203	1	0.4102
Followed By	0.0878	0.0526	0.0522	0.0991	0.1393	0.0854	0.4102	1

Table 2. Correlation analysis for developers

We collected quarterly data (Yamashita et al. 2016) because the average timeframe of minor releases in GitHub is approximately three months. However, our artefact is flexible to handle any duration. We report Pearson correlations of all selected activities for developers (Table 2) and do not find any high correlations. Table 3 illustrates the descriptive statistics of the sample dataset used for evaluation.

Variable	**Description**	**Mean**	**Median**	**Max**	**Min**
Users per Project	Number of users per project	12.60	7	295	4
Projects per user	Number of projects per user	6.92	7	9	5
Commit per user	Number of commits per user	35.77	0	768	0
Commit Comments per user	Number of commit comments per user	0.036	0	1	0
Issues per user	Number of issues per user	0.38	0	10	0

Issue Comments per user	Number of issue comments per user	4.12	0	221	0
Pull Request per user	Number of pull requests per user	13.49	0	492	0
Pull Request Comments per user	Number of pull requests comments per user	1.81	0	120	0

Table 3. Data descriptive statistics (84 developers across 254 projects)

EVALUATION

This study proposes a novel artefact for recommending OSS projects to developers. This artefact is presented in the form of an algorithm and implemented as a Proof-of-Concept (Gregor and Hevner 2013; Iivari 2015) for evaluation purpose, using the real world data. For novel approaches and artefacts, proof-of-concept or prototyping is sufficient to validate the applicability of the proposed solution (Gregor and Hevner 2013). The proof-of-concept prototype of our personalized project rating mechanism is implemented by using MySQL R-DBMS and C# programming language. We have used RecommenderLab R package (Hahsler 2011) to check the accuracy of rating technique. We compared our approach with two general approaches- Popular item recommendation and Random item recommendation, which are available in RecoimmenderLab R package. Next section introduces our evaluation process in depth.

Figure 10 introduces the evaluation process using holdout technique- *Leave-K-Out* (Cremonesi et al. 2008). In this process, randomly selected data-points are removed from OverallRatings matrix. Then we use remaining data points in the OverallRatings matrix to predict the ratings (using RecoimmenderLab R package) that were randomly removed. The error rates are measured by comparing the predicted rates with the actual values. We illustrate this method via an example in Figure 10. Step 1 in Figure 10 shows the OverallRatings matrix. From this matrix, developer B's rating for project L and developer D's rating for project M are randomly removed (see Step 2). We now use the remaining ratings in Table 2, Figure 10, to predict these randomly removed values and compare the predicted ratings with actual ratings to estimate prediction errors. Using 1-NN developers (first nearest neighbourhood) similarity metrics, developer B is similar to C and developer D is also similar to C (see step 3 in Figure 10). The prediction outcome would be developer B's rating on project L as “3” which is equivalent to the actual rating (error =0). For developer D on project M the predicted rate is 5 resulting in the difference of “1” scale from the actual rating (see step 4 in Figure 10).

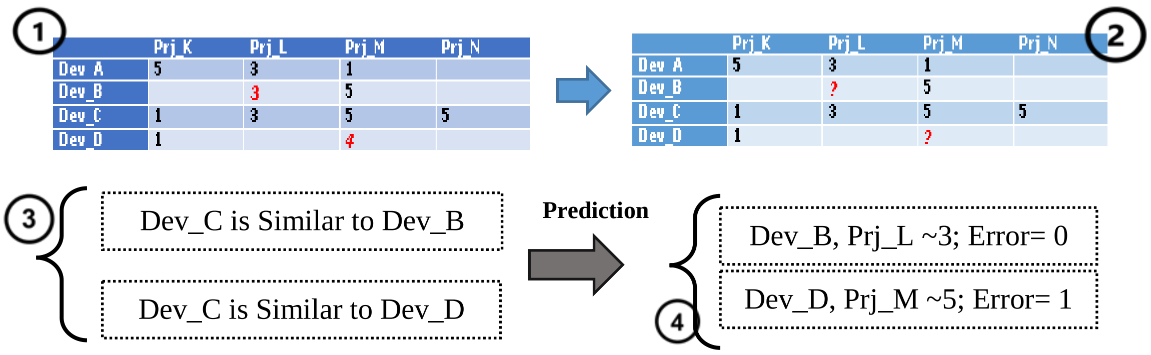


Figure 10. Collaborative filtering evaluation process

In this study, we have compared our proposed moderated OverallRatings with four baseline approaches based on *Average*, *Max*, *Popular* and *Random* computation methods. Note that we computed ActivityRatings matrix for each activity and then by using weights (preferences) for each activity, generated the OverallRatings matrix. In Average approach, we give equal weights to each activity to compute the OverallRatings matrix. The Max approach, considers the maximum rating among all different activities (from ActivityRating matrix) as overall rates. This approach only considers the ratings for the highest weighted activity and ignores the rest. In Popular approach, only most contributed projects are selected to compute OverallRatings. And finally, the Random approach, relies on randomly picked ratings. We analysed the accuracy of our approach by dividing our overall rating matrix into two sets. One for training and one for testing. We used different sizes of training and testing sets to test robustness of our results. We used following combinations of training and testing sets- 1) 70%, and 30%), 2) 80% and 20%), and 3) 90% and 10%.

We ran our test for 100 times to reach the consistent values for our measures. This iterative process is done because the training and testing sets are selected randomly and may show different values based on the sparsity of rating matrix in each division. Following literature in recommender systems (Ricci et al. 2011), we have used RMSE (Root Mean Square Error), MSE (Mean Square Error) and

MAE (Mean Absolute Error) to test the accuracy of our approach. Table 4 shows the average values for each measure on different training and testing set splits.

Table 4. Evaluation result

	Average			**Max**			**Proposed Moderated-Approach**		
	RMSE	*MSE*	*MAE*	*RMSE*	*MSE*	*MAE*	*RMSE*	*MSE*	*MAE*
70-30	0.281	0.098	0.079	0.645	0.471	0.162	0.179	0.044	0.048
80-20	0.307	0.114	0.096	0.736	0.689	0.229	0.185	0.054	0.059
90-10	0.334	0.177	0.135	0.708	0.800	0.262	0.178	0.057	0.070

	Popular			**Random**		
	RMSE	*MSE*	*MAE*	*RMSE*	*MSE*	*MAE*
70-30	0.845	0.772	0.364	1.036	1.104	0.605
80-20	0.848	0.816	0.385	1.075	1.209	0.641
90-10	0.840	0.879	0.413	1.106	1.322	0.683

Figure 11 graphically represents the average values for different approaches we tested and it shows the lowest error rate for our proposed rating technique compare to others approaches. In Figure 11, the first chart on left shows RMSE, middle one illustrates MSE and right one MAE. Each chart shows results for all five approaches, from left to right: *Average, Max, Moderated, Popular, and Random.*

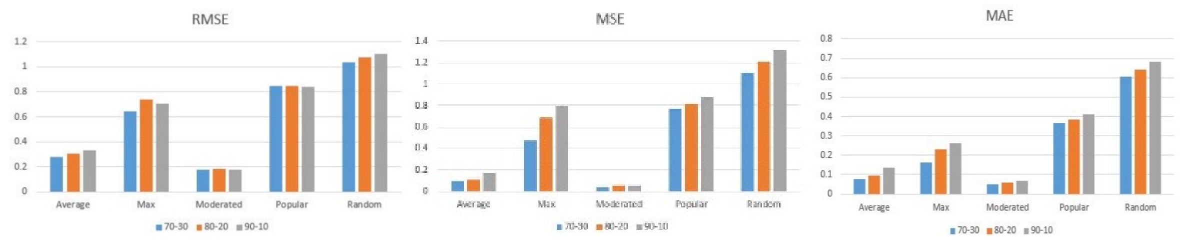


Figure 11. Comparison of different recommendation techniques.

The result of comparison shows that our proposed technique outperforms other approaches. The worst estimation belongs to the *Random* approach and the closer one to our proposed method is the Average approach. It is clear as we increase the size of training set from 70% to 90% the accuracy of predicted values decrease. We suspect that as testing set becomes smaller, (10%) the sparsity of matrix increases and affects the prediction accuracy. Note that our proposed approach does not suffer with this problem. We also observe that though Popular approach doesn't perform better than our proposed approach, the performance of Popular approach is consistent (see Figure 11) for all the training and testing set combinations because one would get almost a similar set of popular projects in the all training-testing set combinations and the popular metric is not a user-dependent metric. Figure 12 shows the boxplots of RMSE, MSE, and MAE for 100 cases captured for (80%, 20%) of each technique. Figure 12 shows the distribution and range of errors for all the evaluation methods. It is clear that our proposed moderated approach performs best among all.

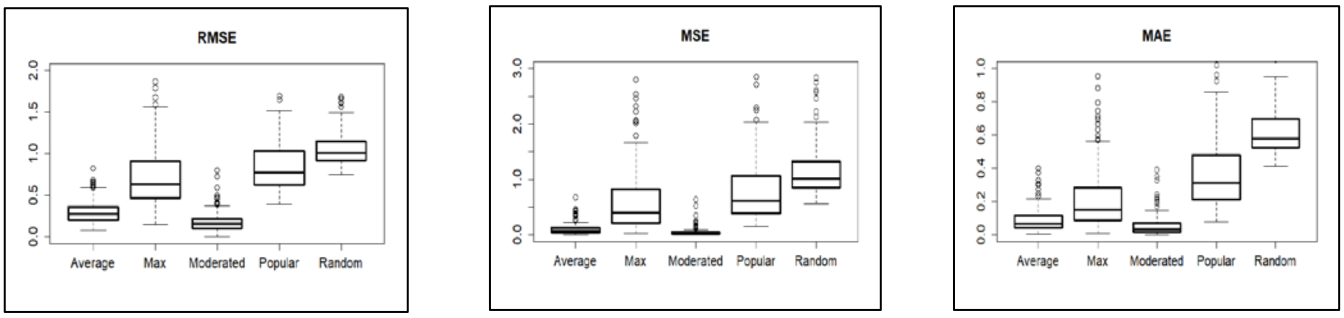


Figure 12. Different techniques comparison from left to right RMSE, MSE, and MAE

Our collaborative filtering method used Z-score for rating matrix normalization, Cosine similarity is used to find the similarity of developers and K-Nearest Neighbourhood is used to find the K similar developers to the focal developer (Hahsler 2011; Ricci et al. 2011).

From the implementation and complexity perspective, our approach is scalable and computationally efficient because the rating algorithm (Algorithm 1) can run in parallel on multiple machines. Running in parallel provides better performance as the numerator and denominator of the Rates function (Equation 7) are independent. Map/Reduce (Shang et al. 2010) is a good choice for processing different parts of Algorithm 1. We can apply Mapper function to calculate activities in different machines and in the reducer sum it up for developers and projects. Note that the *X* matrix is created only once and it just requires updating activities from the most recent time period. Based on the offline creation and calculation of *X* Matrix, this algorithm can perform well in the real-time high-performance recommendation. We can apply the same parallelism techniques for Algorithm 2 as well.

Empirical Evaluation

One pertinent question is what happens if a developer joins the project recommended by our artefact. To answer this question, we examine developers' performance if they join the project recommended by our recommender system.

Sample selection for this evaluation followed a similar criterion as discussed in Data Collection section. As earlier, we focussed on *active developers*, who work on more than five projects, and who joined GitHub in 2012 (Moqri et al. 2018; Nielek et al. 2016). To evaluate our artefact, not only we needed active developers, but also needed projects with enough socio-technical activities of developers, to run our recommendation engine. Note that rating matrix sparsity and cold start are challenging to and collaborative filtering (CF) system (Lam et al. 2008; Lu et al. 2015) and thus are also a limitation of our approach.

We used a sample of 17 active developers associated with 294 OSS projects. OverallRating matrix ranked 7 projects as high ranked and 12 projects as low ranked projects. Recall that developers are recommended to join high ranked projects and not recommended to join low ranked projects. We find that developers who joined high ranked projects, were among top 17% of developers in those projects based on their contributions. Further, in the first year of joining these projects, developers' contribution on these projects was higher compared to developers' total contribution on rest of their projects. In contrast, developers who joined low-ranked projects, were below average in terms of contributions on the project they joined. Further, their overall contribution to these projects was lower than other projects they were associated with. These findings suggest that developers joining our recommended projects is a win-win for both, the developers and the projects.

ROBUSTNESS CHECKS

To test the robustness of our proposed approach, we perform two robustness checks, first by changing the dataset for evaluating the artefact and second by the changing the measurement of socio-technical activities. We describe them next.

It is possible that performance of our method was driven by our sample that was based on developers joining GitHub in a particular month (January 2012). To address this concern, we collected another dataset for developers joining in January 2013 and followed same selection criterion as discussed in Data Collection section. Further, we also changed the time interval for calculating developers' social, technical and collaborative activities. Instead of using quarterly activities of the developers, we used six-monthly data. This dataset resulted into 237developers and 41 projects. We find that though our moderated approach still performed the best, but error rates were slightly higher compared to our results using quarterly data in the earlier dataset. Again, in this case, Average approach was next best after our proposed approach. We don't report results here for brevity.

Recall that to measure developers' socio-technical activities, prior literature has focussed on six activity types- *commit*, *issues* and *pull request* as technical activities and *commit comments*, *pull request comments* and *issue comments* as social activities. Following the literature in software engineering (Xuan and Filkov, 2014), we added *following* and *followed by* as social activities and termed *commit comments, pull request comments* and *issue comments* as collaborative activities. Thus, we expanded the measurement of socio-technical activities (8 activities) to *social*, *technical* and *collaborative* activities (see section Overall Framework). To test the robustness of our proposed approach, we also performed the evaluation of our artefact based on the older measurement of developers' socio-technical activities (using only 6 activities). We find that the error rate was slightly higher; however, it was expected because we ignored two activities (used 6 as opposed to 8). These results underscore the need of using developers' social activities (*following* and *followed by*) to measure their socio-technical activities in OSS projects.

CONCLUSIONS AND FUTURE WORK

Literature in Open source software looks at OSS projects as an outcome of developers' socio-technical activities. Indeed, software development and OSS in particular is an outcome of co-creation which requires lot more than developers' hard technical skills. Recent studies confirm that along with technical skills, developers' social skills are also looked at before they get accepted in OSS projects (Gharehyazie et al. 2015).

Scholars agree that while developers contribute to many OSS projects, they certainly prefer some projects more than others (Hahn et al. 2008; Jarczyk et al. 2014; Tsoy and Staples 2018; van Osch et al. 2011). Further, not only they have different preferences for these projects, there preferences change over time, partly because both developers and OSS projects evolve over time (Happel and Maalej 2008; Robillard et al. 2014). Building on this this literature, we argue that developers' activities/contributions to OSS projects reveal lot more than mere associations with these projects. We argue that a detailed look at their activities/contributions on OSS projects can unlock the secret and can reveal the extent and evolving nature of their preferences for projects, based on, how much, how often and what they contribute on these projects. We address these issues by developing an artefact that can recommend projects to developers as we discuss next.

This study presents a framework for developing a personalized OSS project ranking mechanism for developers. Our framework proposes to use developers' historical activities/contributions on social coding platforms such as GitHub, to create an implicit feedback rating mechanism for the projects they contributed. Developers' activities on these social coding platforms are transparent and therefore support the viability of the proposed framework and ranking mechanism. Since these rankings are personalized for each developer based on their contributions, these project rankings are then used for recommending OSS projects to developers using collaborative filtering (CF) technique. This recommender system is expected to match interested developers to projects in a way to reduce developers' attrition (Steinmacher et al. 2015) and project failure (Schilling 2012) which has threatened the growth and sustainability of OSS ecosystem in recent years.

The OSS project recommendation that we propose, is a multi-criteria decision-making problem. The project recommendation is built on developers' interest/contribution in social, collaborative and technical activities. On one side, we considered the temporal changes in developers' contributions over time, in terms of projects and activity types. And, on the other side, we considered the changes in projects' internal activities over time.

Overall, this study makes two main contributions to the literature. First, we propose to categorize developers' activities/contributions as *Social, Technical and Collaborative* activities. Existing work categorizes these activities as either social or technical. We add a new category- collaborative activities. Prior work has discussed the importance of developers' socio-technical activities in OSS ecosystem but literature is scattered with often contradictory findings/recommendations on categorization of these activities (Moqri et al. 2018; Sarker et al. 2019). We propose a coherent categorization of these activities based on literature in software engineering.

Second, we develop an artefact to generate personalized project rankings for developers. Since developers do not directly rate projects, we use developers' historical contributions via social, technical and collaborative activities across different projects to capture their preferences and create implicit feedback ratings. Ratings are moderated based on the proposed multi-criteria decision making approach to compute the overall moderated ratings of projects. The moderated overall ratings are an input to a collaborative filtering recommender system. As a proof-of-concept, the proposed approach is evaluated based on a real longitudinal dataset of OSS projects and developers hosted on GitHub. We compared the proposed rating method with other possible state of the art approaches, and find that our approach shows high accuracy in term of distance between real and predicted data. Our contributions are presented in the format of three algorithms and implemented as a prototype application.

However, similar to other studies this research has some limitations, which are discussed next. First, the dataset comprises of developers' activities only on GitHub while developers may have contributed to projects hosted on Bitbucket, GitLab, SourceForge or other repositories. Although adding these repositories may provide a more complete dataset of developers' activities, it may not affect this study for following reasons. 1) A change in the dataset does not affect the design our project ranking mechanism, the main contribution of this study, 2) GitHub is by far the largest repository, which means, GitHub data can provide dominant patterns in the OSS community, 3). We have focused on GitHub data since 2012, when it was accepted as the most matured OSS repository (Moqri et al. 2018; Nielek et al. 2016). Further, the prospect of using a third party data integrator such as OpenHub (Ohloh) is not helpful because Ohloh only collects data on a small number of OSS projects which is not comparable with size and scale of GitHub and not all the GitHub contributors are listed on Ohloh and therefore, GitHub data has been used in many studies (Cosentino et al. 2017; Sarker et al. 2019).

Second, Collaborative filtering is sensitive to the matrix sparsity and as we have used the general collaborative filtering technique in this study, this study may also suffer from this problem. Matrix factorization and model-based techniques such as clustering (Ricci et al. 2011) may help in this situation. We need to evaluate the quality of proposed rating approach in the sparse cases with the mentioned techniques.

Third, similar to other pure user-based collaborative filtering technique this artefact may face cold start problems with (Lam et al. 2008) new projects and developers with the low historical background. Future research aims to use the combination of knowledge-based recommender systems and model-based techniques to address this limitation.

Fourth, we have evaluated our proposed model with the limited number of GitHub projects and developers as a proof-of-concept. We only selected developers who contributed to at least 5 projects to overcome the cold-start and sparsity issues of collaborative filtering. However, cold-start and sparsity are always challenging in any collaborative filtering based recommendation system (Adomavicius and Tuzhilin 2005). Further, we argue that using this same dataset, our approach outperformed other baseline approaches.

Fifth, in recommending projects to developers, we didn’t account for developers’ preferences in programming languages, topics, licences and prior ties, etc. We argue that we indirectly account for these by using collaborative filtering approach.

And finally, the sample size used for the proof-of-concept may be small, however, it is comparable to other studies (Blincoe, et al., 2016). Note, that our dataset is small because the period of data collection is also restricted. To address this issue we have evaluated our artefact with other datasets to validate our proposed technique.

In summary, we argue that while extant literature has investigated how developers' prior ties with other developers affect their decision to join an OSS project (Hahn et al. 2008), we add to that literature and develop a recommender system that recommends projects considering developers' preferences and projects' goals beyond developers' ties with other developers. In future research, we aim to merge this approach with knowledge based and content-based techniques to develop a hybrid recommender system. Future work can also use social connectivity to measure similarities among developers. Another possible extension to the current method would be applying model-based technique such as clustering on developers to reduce the matrix sparsity and cold-start problem.

REFERENCES

Adomavicius, G., and Kwon, Y. 2015. "Multi-Criteria Recommender Systems," in *Recommender Systems Handbook*. Springer, pp. 847-880.

Adomavicius, G., and Tuzhilin, A. 2005. "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE transactions on knowledge and data engineering* (17:6), pp. 734-749.

Alexy, O., and Leitner, M. 2011. "A Fistful of Dollars: Are Financial Rewards a Suitable Management Practice for Distributed Models of Innovation?," *European Management Review* (8:3), pp. 165-185.

Allaho, M. Y., and Lee, W.-C. 2013. "Analyzing the Social Ties and Structure of Contributors in Open Source Software Community," *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*: IEEE, pp. 56-60.

Allaho, M. Y., and Lee, W.-C. 2014. "Increasing the Responsiveness of Recommended Expert Collaborators for Online Open Projects," *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*: ACM, pp. 749-758.

Anvik, J., Hiew, L., and Murphy, G. C. 2006. "Who Should Fix This Bug?," *Proceedings of the 28th international conference on Software engineering*: ACM, pp. 361-370.

Anvik, J., and Murphy, G. C. 2011. "Reducing the Effort of Bug Report Triage: Recommenders for Development-Oriented Decisions," *ACM Transactions on Software Engineering and Methodology (TOSEM)* (20:3), p. 10.

Badashian, A. S., Esteki, A., Gholipour, A., Hindle, A., and Stroulia, E. 2014. "Involvement, Contribution and Influence in Github and Stack Overflow," *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*: IBM Corp., pp. 19-33.

Badashian, A. S., Hindle, A., and Stroulia, E. 2015. "Crowdsourced Bug Triaging," *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*: IEEE, pp. 506-510.

Badashian, A. S., and Stroulia, E. 2016. "Measuring User Influence in Github: The Million Follower Fallacy," *CrowdSourcing in Software Engineering (CSI-SE), 2016 IEEE/ACM 3rd International Workshop on*: IEEE, pp. 15-21.

Bayati, S. 2018. "Understanding Newcomers Success in Open Source Community," *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*: ACM, pp. 224-225.

Bayati, S., and Peiris, K. 2018. "Road to Success: How Newcomers Gain Reputation in Open Source Community," in: *PACIS 2018*. Japan.

Bird, C., Nagappan, N., Gall, H., Murphy, B., and Devanbu, P. 2009. "Putting It All Together: Using Socio-Technical Networks to Predict Failures," *Software Reliability Engineering, 2009. ISSRE'09. 20th International Symposium on*: IEEE, pp. 109-119.

Bissyandé, T. F., Lo, D., Jiang, L., Reveillere, L., Klein, J., and Le Traon, Y. 2013. "Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from Github," *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*: IEEE, pp. 188-197.

Blincoe, K., Sheoran, J., Goggins, S., Petakovic, E., and Damian, D. 2016. "Understanding the Popular Users: Following, Affiliation Influence and Leadership on Github," *Information and Software Technology* (70), pp. 30-39.

Carillo, K., Huff, S., and Chawner, B. 2017. "What Makes a Good Contributor? Understanding Contributor Behavior within Large Free/Open Source Software Projects–a Socialization Perspective," *The Journal of Strategic Information Systems* (26:4), pp. 322-359.

Carillo, K. D. A., Marsan, J., and Negoita, B. 2016. "Towards Developing a Theory of Toxicity in the Context of Free/Open Source Software & Peer Production Communities," in: *SIGOPEN 2016*.

Cha, M., Haddadi, H., Benevenuto, F., and Gummadi, P. K. 2010. "Measuring User Influence in Twitter: The Million Follower Fallacy," *Icwsm* (10:10-17), p. 30.

Chengalur-Smith, I., Sidorova, A., and Daniel, S. 2010. "Sustainability of Free/Libre Open Source Projects: A Longitudinal Study," *Journal of the Association for Information Systems* (11:11), p. 657.

Choi, K., Yoo, D., Kim, G., and Suh, Y. 2012. "A Hybrid Online-Product Recommendation System: Combining Implicit Rating-Based Collaborative Filtering and Sequential Pattern Analysis," *electronic commerce research and applications* (11:4), pp. 309-317.

Chou, S. W., and He, M. Y. 2011. "The Factors That Affect the Performance of Open Source Software Development–the Perspective of Social Capital and Expertise Integration," *Information Systems Journal* (21:2), pp. 195-219.

Cosentino, V., Izquierdo, J. L. C., and Cabot, J. 2017. "A Systematic Mapping Study of Software Development with Github," *IEEE Access* (5), pp. 7173-7192.

Cremonesi, P., Turrin, R., Lentini, E., and Matteucci, M. 2008. "An Evaluation Methodology for Collaborative Recommender Systems," *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*: IEEE, pp. 224-231.

Crowston, K., and Howison, J. 2005. "The Social Structure of Free and Open Source Software Development," *First Monday* (10:2).

Crowston, K., Howison, J., and Annabi, H. 2006. "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice* (11:2), pp. 123-148.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. 2012. "Social Coding in Github: Transparency and Collaboration in an Open Software Repository," *Proceedings of the ACM 2012 conference on computer supported cooperative work*: ACM, pp. 1277-1286.

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. 2013. "Leveraging Transparency," *IEEE software* (30:1), pp. 37-43.

Daniel, S., Midha, V., Bhattacherhjee, A., and Singh, S. P. 2018. "Sourcing Knowledge in Open Source Software Projects: The Impacts of Internal and External Social Capital on Project Success," *The Journal of Strategic Information Systems* (27:3), pp. 237-256.

Daniel, S., and Stewart, K. 2016. "Open Source Project Success: Resource Access, Flow, and Integration," *The Journal of Strategic Information Systems* (25:3), pp. 159-176.

De Vries, L., Gensler, S., and Leeflang, P. S. 2012. "Popularity of Brand Posts on Brand Fan Pages: An Investigation of the Effects of Social Media Marketing," *Journal of interactive marketing* (26:2), pp. 83-91.

Erickson, T., and Kellogg, W. A. 2000. "Social Translucence: An Approach to Designing Systems That Support Social Processes," *ACM transactions on computer-human interaction (TOCHI)* (7:1), pp. 59-83.

Fang, Y., and Neufeld, D. 2009. "Understanding Sustained Participation in Open Source Software Projects," *Journal of Management Information Systems* (25:4), pp. 9-50.

Fronchetti, F., Wiese, I., Pinto, G., and Steinmacher, I. 2019. "What Attracts Newcomers to Onboard on Oss Projects? Tl; Dr: Popularity," *IFIP International Conference on Open Source Systems*: Springer, pp. 91-103.

Gharehyazie, M., Posnett, D., Vasilescu, B., and Filkov, V. 2015. "Developer Initiation and Social Interactions in Oss: A Case Study of the Apache Software Foundation," *Empirical Software Engineering* (20:5), pp. 1318-1353.

Goggins, S., and Petakovic, E. 2014. "Connecting Theory to Social Technology Platforms: A Framework for Measuring Influence in Context," *American Behavioral Scientist* (58:10), pp. 1376-1392.

Gousios, G. 2013. "The Ghtorent Dataset and Tool Suite," *Proceedings of the 10th Working Conference on Mining Software Repositories*: IEEE Press, pp. 233-236.

Gousios, G., Storey, M.-A., and Bacchelli, A. 2016. "Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective," *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*: IEEE, pp. 285-296.

Gregor, S., and Hevner, A. R. 2013. "Positioning and Presenting Design Science Research for Maximum Impact," *MIS quarterly* (37:2), pp. 337-355.

Grewal, R., Lilien, G. L., and Mallapragada, G. 2006. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management Science* (52:7), pp. 1043-1056.

Grigorik, I. 2012. "The Github Archive." Mar.

Guzman, E., Azócar, D., and Li, Y. 2014. "Sentiment Analysis of Commit Comments in Github: An Empirical Study," *Proceedings of the 11th Working Conference on Mining Software Repositories*: ACM, pp. 352-355.

Guzzi, A., Bacchelli, A., Lanza, M., Pinzger, M., and Van Deursen, A. 2013. "Communication in Open Source Software Development Mailing Lists," *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*: IEEE, pp. 277-286.

Hahn, J., Moon, J. Y., and Zhang, C. 2006. "Impact of Social Ties on Open Source Project Team Formation," *IFIP International Conference on Open Source Systems*: Springer, pp. 307-317.

Hahn, J., Moon, J. Y., and Zhang, C. 2008. "Emergence of New Project Teams from Open Source Software Developer Networks: Impact of Prior Collaboration Ties," *Information Systems Research* (19:3), pp. 369-391.

Hahsler, M. 2011. "Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms," *Southern Methodist University*).

Hannebauer, C., and Gruhn, V. 2017. "On the Relationship between Newcomer Motivations and Contribution Barriers in Open Source Projects," *Proceedings of the 13th International Symposium on Open Collaboration*: ACM, p. 2.

Happel, H.-J., and Maalej, W. 2008. "Potentials and Challenges of Recommendation Systems for Software Development," *Proceedings of the 2008 international workshop on Recommendation systems for software engineering*: ACM, pp. 11-15.

Hevner, A., Salvatore, M., Jinsoo, P., and Sudha, R. 2004. "Design Science in Information Systems Research," *MIS quarterly* (28:1), pp. 75-105.

Holmes, R., and Murphy, G. C. 2005. "Using Structural Context to Recommend Source Code Examples," *Proceedings of the 27th international conference on Software engineering*: ACM, pp. 117-125.

Hu, Y., Koren, Y., and Volinsky, C. 2008. "Collaborative Filtering for Implicit Feedback Datasets," *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*: Ieee, pp. 263-272.

Iivari, J. 2015. "Distinguishing and Contrasting Two Strategies for Design Science Research," *European Journal of Information Systems* (24:1), pp. 107-115.

Jarczyk, O., Gruszka, B., Jaroszewicz, S., Bukowski, L., and Wierzbicki, A. 2014. "Github Projects. Quality Analysis of Open-Source Software," in *Social Informatics*. Springer, pp. 80-94.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., and Zhang, L. 2016. "Why and How Developers Fork What from Whom in Github," *Empirical Software Engineering*), pp. 1-32.

Jurado, F., and Rodriguez, P. 2015. "Sentiment Analysis in Monitoring Software Development Processes: An Exploratory Case Study on Github's Project Issues," *Journal of Systems and Software* (104), pp. 82-89.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. 2014. "The Promises and Perils of Mining Github," *Proceedings of the 11th working conference on mining software repositories*: ACM, pp. 92-101.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. 2016. "An in-Depth Study of the Promises and Perils of Mining Github," *Empirical Software Engineering* (21:5), pp. 2035-2071.

Kelly, D., and Teevan, J. 2003. "Implicit Feedback for Inferring User Preference: A Bibliography," *Acm Sigir Forum*: ACM, pp. 18-28.

Krishnamurthy, S., Ou, S., and Tripathi, A. K. 2014. "Acceptance of Monetary Rewards in Open Source Software Development," *Research Policy* (43:4), pp. 632-644.

Lam, X. N., Vu, T., Le, T. D., and Duong, A. D. 2008. "Addressing Cold-Start Problem in Recommendation Systems," *Proceedings of the 2nd international conference on Ubiquitous information management and communication*: ACM, pp. 208-211.

Lee, M. J., Ferwerda, B., Choi, J., Hahn, J., Moon, J. Y., and Kim, J. 2013. "Github Developers Use Rockstars to Overcome Overflow of News," *CHI'13 Extended Abstracts on Human Factors in Computing Systems*: ACM, pp. 133-138.

Lee, S. K., Cho, Y. H., and Kim, S. H. 2010. "Collaborative Filtering with Ordinal Scale-Based Implicit Ratings for Mobile Music Recommendations," *Information Sciences* (180:11), pp. 2142-2155.

Lee, T. Q., Park, Y., and Park, Y.-T. 2008. "A Time-Based Approach to Effective Recommender Systems Using Implicit Feedback," *Expert systems with applications* (34:4), pp. 3055-3062.

Lu, J., Wu, D., Mao, M., Wang, W., and Zhang, G. 2015. "Recommender System Application Developments: A Survey," *Decision Support Systems* (74), pp. 12-32.

Maalej, W., and Thurimella, A. K. 2009. "Towards a Research Agenda for Recommendation Systems in Requirements Engineering," *Second International Workshop on Managing Requirements Knowledge*.

Montaner, M., López, B., and de la Rosa, J. L. 2002. "Opinion-Based Filtering through Trust," *International Workshop on Cooperative Information Agents*: Springer, pp. 164-178.

Moqri, M., Mei, X., Qiu, L., and Bandyopadhyay, S. 2018. "Effect of “Following” on Contributions to Open Source Communities," *Journal of Management Information Systems* (35:4), pp. 1188-1217.

Naguib, H., Narayan, N., Brugge, B., and Helal, D. 2013. "Bug Report Assignee Recommendation Using Activity Profiles," *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*: IEEE, pp. 22-30.

Nielek, R., Jarczyk, O., Pawlak, K., Bukowski, L., Bartusiak, R., and Wierzbicki, A. 2016. "Choose a Job You Love: Predicting Choices of Github Developers," *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*: IEEE, pp. 200-207.

Oh, W., Moon, J. Y., Hahn, J., and Kim, T. 2016. "Research Note—Leader Influence on Sustained Participation in Online Collaborative Work Communities: A Simulation-Based Approach," *Information Systems Research* (27:2), pp. 383-402.

Palanivel, K., and Sivakumar, R. 2010. "A Study on Implicit Feedback in Multicriteria E-Commerce Recommender System," *Journal of Electronic Commerce Research* (11:2), p. 140.

Palma, F., Farzin, H., Gueheneuc, Y.-G., and Moha, N. 2012. "Recommendation System for Design Patterns in Software Development: An Dpr Overview," *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*: IEEE Press, pp. 1-5.

Paulson, J. W., Succi, G., and Eberlein, A. 2004. "An Empirical Study of Open-Source and Closed-Source Software Products," *IEEE Transactions on Software Engineering* (30:4), pp. 246-256.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. 2007. "A Design Science Research Methodology for Information Systems Research," *Journal of management information systems* (24:3), pp. 45-77.

Pletea, D., Vasilescu, B., and Serebrenik, A. 2014. "Security and Emotion: Sentiment Analysis of Security Discussions on Github," *Proceedings of the 11th working conference on mining software repositories*: ACM, pp. 348-351.

Qureshi, I., and Fang, Y. 2011. "Socialization in Open Source Software Projects: A Growth Mixture Modeling Approach," *Organizational Research Methods* (14:1), pp. 208-238.

Ricci, F., Rokach, L., and Shapira, B. 2011. *Introduction to Recommender Systems Handbook*. Springer.

Robbes, R., and Röthlisberger, D. 2013. "Using Developer Interaction Data to Compare Expertise Metrics," *Proceedings of the 10th Working Conference on Mining Software Repositories*: IEEE Press, pp. 297-300.

Roberts, J. A., Hann, I.-H., and Slaughter, S. A. 2006. "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management science* (52:7), pp. 984-999.

Robillard, M. P., Maalej, W., Walker, R. J., and Zimmermann, T. 2014. *Recommendation Systems in Software Engineering*. Springer.

Robillard, M. P., Walker, R. J., and Zimmermann, T. 2010. "Recommendation Systems for Software Engineering," *Software, IEEE* (27:4), pp. 80-86.

Sarker, F., Vasilescu, B., Blincoe, K., and Filkov, V. 2019. "Socio-Technical Work-Rate Increase Associates with Changes in Work Patterns in Online Projects," in: *ICSE 2019*. Canada.

Schafer, J. B., Konstan, J., and Riedl, J. 1999. "Recommender Systems in E-Commerce," *Proceedings of the 1st ACM conference on Electronic commerce*: ACM, pp. 158-166.

Schall, D. 2014. "Who to Follow Recommendation in Large-Scale Online Development Communities," *Information and Software Technology* (56:12), pp. 1543-1555.

Schilling, A. 2012. "Links to the Source-a Multidimensional View of Social Ties for the Retention of Floss Developers," *Proceedings of the 50th annual conference on Computers and People Research*: ACM, pp. 103-108.

Schilling, A. 2014. "What Do We Know About Floss Developers' Attraction, Retention, and Commitment? A Literature Review," *System Sciences (HICSS), 2014 47th Hawaii International Conference on*: IEEE, pp. 4003-4012.

Schilling, A., Laumer, S., and Weitzel, T. 2012a. "Train and Retain: The Impact of Mentoring on the Retention of Floss Developers," *Proceedings of the 50th annual conference on Computers and People Research*: ACM, pp. 79-84.

Schilling, A., Laumer, S., and Weitzel, T. 2012b. "Who Will Remain? An Evaluation of Actual Person-Job and Person-Team Fit to Predict Developer Retention in Floss Projects," *System Science (HICSS), 2012 45th Hawaii International Conference on*: IEEE, pp. 3446-3455.

Schilling, A., Laumer, S., and Weitzel, T. 2013. "Together but Apart: How Spatial, Temporal and Cultural Distances Affect Floss Developers' Project Retention," *Proceedings of the 2013 annual conference on Computers and people research*: ACM, pp. 167-172.

Shah, S. K. 2006. "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management science* (52:7), pp. 1000-1014.

Shang, W., Adams, B., and Hassan, A. E. 2010. "An Experience Report on Scaling Tools for Mining Software Repositories Using Mapreduce," *Proceedings of the IEEE/ACM international conference on Automated software engineering*: ACM, pp. 275-284.

Sharma, P. N., Hulland, J., and Daniel, S. 2012. "Examining Turnover in Open Source Software Projects Using Logistic Hierarchical Linear Modeling Approach," *IFIP International Conference on Open Source Systems*: Springer, pp. 331-337.

Shokripour, R., Anvik, J., Kasirun, Z. M., and Zamani, S. 2013. "Why So Complicated? Simple Term Filtering and Weighting for Location-Based Bug Report Assignment Recommendation," *Proceedings of the 10th Working Conference on Mining Software Repositories*: IEEE Press, pp. 2-11.

Singh, P. V., Tan, Y., and Mookerjee, V. 2011. "Network Effects: The Influence of Structural Capital on Open Source Project Success," *Mis Quarterly*), pp. 813-829.

Steinmacher, I., Pinto, G., Wiese, I. S., and Gerosa, M. A. 2018. "Almost There: A Study on Quasi-Contributors in Open-Source Software Projects," *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*: IEEE, pp. 256-266.

Steinmacher, I., Silva, M. A. G., Gerosa, M. A., and Redmiles, D. F. 2015. "A Systematic Literature Review on the Barriers Faced by Newcomers to Open Source Software Projects," *Information and Software Technology* (59), pp. 67-85.

Swani, K., Milne, G. R., Brown, B. P., Assaf, A. G., and Donthu, N. 2017. "What Messages to Post? Evaluating the Popularity of Social Media Communications in Business Versus Consumer Markets," *Industrial Marketing Management* (62), pp. 77-87.

Terceiro, A., Souza, R., and Chavez, C. 2012. "Patterns for Engagement in Free Software Projects," *Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming*: ACM, p. 3.

Tsay, J., Dabbish, L., and Herbsleb, J. 2014a. "Influence of Social and Technical Factors for Evaluating Contribution in Github," *Proceedings of the 36th international conference on Software engineering*: ACM, pp. 356-366.

Tsay, J., Dabbish, L., and Herbsleb, J. 2014b. "Let's Talk About It: Evaluating Contributions through Discussion in Github," *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*: ACM, pp. 144-154.

Tsoy, M., and Staples, D. S. 2018. "Role of Reputation Cues in Trust Formation for a Developer's Decision to Join Open Source Software Projects,").

van Osch, W., Adelaar, T., and Pith, M. 2011. "So Many Developers, So Many Projects: Toward a Motivation-Based Theory of Project Selection," *AMCIS*.

Vasilescu, B., Blincoe, K., Xuan, Q., Casalnuovo, C., Damian, D., Devanbu, P., and Filkov, V. 2016. "The Sky Is Not the Limit: Multitasking across Github Projects," *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*: IEEE, pp. 994-1005.

Vasilescu, B., Filkov, V., and Serebrenik, A. 2013. "Stackoverflow and Github: Associations between Software Development and Crowdsourced Knowledge," *Social Computing (SocialCom), 2013 International Conference on*: IEEE, pp. 188-195.

Vasilescu, B., Serebrenik, A., and Filkov, V. 2015. "A Data Set for Social Diversity Studies of Github Teams," *Proceedings of the 12th Working Conference on Mining Software Repositories*: IEEE Press, pp. 514-517.

Von Krogh, G., Spaeth, S., and Lakhani, K. R. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy* (32:7), pp. 1217-1241.

Von Krogh, G., and Von Hippel, E. 2006. "The Promise of Research on Open Source Software," *Management science* (52:7), pp. 975-983.

Wang, H., Guo, X., Zhang, M., Wei, Q., and Chen, G. 2015. "Predicting the Incremental Benefits of Online Information Search for Heterogeneous Consumers," *Decision Sciences*).

Wu, J., and Goh, K. Y. 2009. "Evaluating Longitudinal Success of Open Source Software Projects: A Social Network Perspective," *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*: IEEE, pp. 1-10.

Wu, Y., Kropczynski, J., Shih, P. C., and Carroll, J. M. 2014. "Exploring the Ecosystem of Software Developers on Github and Other Platforms," *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*: ACM, pp. 265-268.

Xie, T., Thummalapenta, S., Lo, D., and Liu, C. 2009. "Data Mining for Software Engineering," *Computer*:8), pp. 55-62.

Yamashita, K., Kamei, Y., McIntosh, S., Hassan, A. E., and Ubayashi, N. 2016. "Magnet or Sticky? Measuring Project Characteristics from the Perspective of Developer Attraction and Retention," *Journal of Information Processing* (24:2), pp. 339-348.

Yang, X., Guo, Y., Liu, Y., and Steck, H. 2014. "A Survey of Collaborative Filtering Based Social Recommender Systems," *Computer Communications* (41), pp. 1-10.

Yu, Y., Benlian, A., and Hess, T. 2012. "An Empirical Study of Volunteer Members' Perceived Turnover in Open Source Software Projects," *System Science (HICSS), 2012 45th Hawaii International Conference on*: IEEE, pp. 3396-3405.

Yu, Y., Wang, H., Yin, G., and Wang, T. 2016. "Reviewer Recommendation for Pull-Requests in Github: What Can We Learn from Code Review and Bug Assignment?," *Information and Software Technology*).

Zimmermann, T., Dallmeier, V., Halachev, K., and Zeller, A. 2005. "Erose: Guiding Programmers in Eclipse," *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*: ACM, pp. 186-187.