11-2017

Understanding inactive yet available assignees in GitHub

Jing JIANG

David LO
Singapore Management University, davidlo@smu.edu.sg

Xinyu MA

Fuli FENG

Li ZHANG

DOI: https://doi.org/10.1016/j.infsof.2017.06.005

Citation

JIANG, Jing; David LO; MA, Xinyu; FENG, Fuli; and ZHANG, Li. Understanding inactive yet available assignees in GitHub. (2017). *Information and Software Technology*. 91, 44-55. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/3785

Published in Information and Software Technology, 2017 November, Volume 91, Pages 44-55
http://doi.org/10.1016/j.infsof.2017.06.005

Understanding inactive yet available assignees in GitHub

Jing Jiang[a], David Lo[b], Xinyu Ma[a], Fuli Feng[a], Li Zhang[a,*]

[a] *State Key Laboratory of Software Development Environment, Beihang University, Beijing, China*
[b] *School of Information Systems, Singapore Management University, Singapore*

A R T I C L E I N F O

Article history:
Received 29 June 2016
Revised 10 June 2017
Accepted 21 June 2017
Available online 22 June 2017

Keywords:
Assignee
Inactive
GitHub

A B S T R A C T

Context: In GitHub, an issue or a pull request can be assigned to a specific *assignee* who is responsible for working on this issue or pull request. Due to the principle of voluntary participation, available assignees may remain inactive in projects. If assignees ever participate in projects, they are *active assignees*; otherwise, they are *inactive yet available assignees* (inactive assignees for short).
Objective: Our objective in this paper is to provide a comprehensive analysis of inactive yet available assignees in GitHub.
Method: We collect 2,374,474 records of activities in 37 popular projects, and 797,756 records of activities in 687 projects belonging to 8 organizations. We compute the percentage of inactive assignees in projects, and compare projects with and without inactive assignees. Then we analyze datasets to explore why some assignees are inactive. Finally, we send questionnaires to understand impacts of inactive assignees.
Results: We find that some projects have high percentage of inactive yet available assignees. For example, 66.35% of assignees never participate in the project *paperclip*. The project *paperclip* belongs to the organization *thoughtbot*. In the organization *thoughtbot*, 84.4% of projects have more than 80% of inactive assignees. We further observe that the main reason for developers being inactive assignees is that developers work for organizations and automatically become available assignees of some projects in the organizations. However, these developers do not work on projects. 37.25% of developers that we have surveyed agree that inactive assignees affect open source software development (i.e., causing unresolved issues or pull requests, and delaying software development).
Conclusion: Some organizations should improve team management, and carefully select developers to become assignees in projects. Future studies about assignees should be careful to perform data cleaning, since some available assignees are added by virtue of their employment and do not really work on projects.

1. Introduction

GitHub[1] is a web-based hosting service for software development repositories and one of the largest and most popular open source communities in the world [1]. GitHub implements an in-house issue tracking system where developers file issues [2]. Issues are a great way to keep track of tasks, enhancements, and bugs for projects. GitHub supports the pull-based software development for code contribution [3,4]. Pull requests are submitted when developers want to merge their code changes into original projects.

In GitHub, an issue or a pull request can be assigned to a specific *assignee* who is responsible for working on this issue or pull request [2]. Developers write issue reports to discuss bugs or feature requests [2], while they open pull requests to submit code changes which they want to merge into original projects [5]. GitHub allows repositories to identify a set of developers who can be assigned to issues and pull requests. In this paper, we refer to this set of developers as *available assignees* (assignees for short). OSS (open source software) teams tend to be self-organized, fluid and diverse, resulting in high turnover [6]. Due to the principle of voluntary participation [7], assignees have the freedom to decide their activities and even remain inactive in projects. If assignees ever participate in projects, they are *active assignees*; otherwise, they are *inactive yet available assignees* (inactive assignees for short).

Previous work found that assignees had more followers than reporters who created issues [8]. Some studies designed approaches

* Corresponding author.
E-mail addresses: jiangjing@buaa.edu.cn (J. Jiang), davidlo@smu.edu.sg (D. Lo), sdxyzlc@126.com (X. Ma), fulifeng93@gmail.com (F. Feng), lily@buaa.edu.cn (L. Zhang).

[1] http://github.com.

[2] https://help.github.com/articles/assigning-issues-and-pull-requests-to-other-github-users/.

to recommend reviewers who were responsible for working on pull requests [9–11]. However, no works have studied the activeness of all available assignees. It remains unknown whether assignees are inactive, why some available assignees are inactive, and how inactive assignees affect software development. It is important to provide a comprehensive analysis of inactive yet available assignees in GitHub.

In this work, we analyze inactive yet available assignees in GitHub. GitHub is a web-based hosting service for software development repositories. It is one of the world's largest code hosting sites. For our analysis, we use GitHub API to collect 2,374,474 records of activities in 37 popular projects, and 797,756 records of activities in 687 projects belonging to 8 organizations (Section 3). Then we analyze 37 popular projects, study the percentage of inactive assignees in projects, and compare projects with and without inactive assignees (Section 4). Next, we study 17 projects with inactive assignees and 687 projects belonging to 8 organizations, and explore why some assignees are inactive but they become available assignees (Section 5). Finally, we study impacts of inactive assignees on software development (Section 6). Our study provides a number of insights into inactive assignees in GitHub.

- Some projects have high percentage of inactive yet available assignees. For example, 66.35% of assignees never participate in project *paperclip*[3]. The project *paperclip* belongs to the organization *thoughtbot*. In the organization *thoughtbot*, 84.4% of projects have more than 80% of inactive assignees.
- The main reason for developers being inactive assignees is that developers work for organizations and automatically become available assignees of some projects in the organizations. However, these developers do not work on projects.
- 37.25% of developers that we have surveyed agree that inactive assignees affect open source software development (i.e., causing unresolved issues or pull requests, and delaying software development).

These findings imply that some organizations should improve team management, and carefully select developers to become assignees in projects. In some organizations' projects, available assignees include developers who never participate in projects but work for the respective owner organizations. It causes the high percentage of inactive assignees, to whom issues or pull requests may be assigned. If issues or pull requests are assigned to inactive assignees, these issues or pull requests will not be handled by these assignees and remain open for a long time, which may discourage submitters of issues or pull requests. Reviewer recommendation should exclude inactive yet available assignees for pull requests in GitHub. Future studies about assignees should be careful to perform data cleaning, since some available assignees are added by virtue of their employment and do not really work on projects.

2. Background and research questions

In this section, we firstly provide background information about development activities in GitHub. Then we introduce research questions.

2.1. Background

GitHub is a web-based hosting service for software development projects. It has become one of the world's largest open source communities. In GitHub, an issue or a pull request can be assigned to a specific developer who is responsible for working on this issue or pull request. Available assignees include project owner, collaborators on personal projects, or members of organization with read permissions on the project. Assignees may participate in open source software projects through different ways. We introduce various kinds of activities as follow.

Issue Herzig et al. observed that some issues described new feature requests, rather than bugs [12]. Developers also write issue reports to identify bugs, document software codes, and enhance the software via feature requests in GitHub [2]. We consider all issues, and do not distinguish issues by their functions. This is because activities in any kinds of issues all show the activeness of assignees. An issue can be assigned to a specific assignee who is responsible for working on the issue. Assignees may exchange comments, provide feedback, or discuss problems mentioned in issues. Assignees may also report issues and give suggestions for the improvement of projects.

Pull request Contributors fork projects and make changes to implement new features or fix bugs. Contributors submit pull requests when they want to merge code changes into the main project. Pull requests may be assigned to specific assignees. Assignees inspect code changes, evaluate potential contributions, and decide whether to accept pull requests or not [3]. Assignees sometimes ask contributors to make updates and submit new commits for re-evaluation. Assignees and other interested users exchange comments, perhaps to suggest improvements or negotiate over code changes. Assignees may also create pull requests to submit codes.

Commit Some pull requests may be assigned to specific assignees. Assignees evaluate code changes submitted by pull requests, and merge satisfactory code changes into projects. Assignees may commit changes which originate from other developers. Assignees may also exchange comments and discuss commits.

GitHub displays developers' contributions in their profiles, which mainly considers their activities in issues, pull requests and commits[4]. In this paper, we also mainly study assignees' activities in pull requests, issues and commits. *Assignees are considered as active, if assignees participate in issues, pull requests or commits; otherwise, assignees are considered as inactive.* Issues, pull requests and commits are directly associated with the development of OSS projects. Other activities are not considered in this work. For example, assignees may watch projects and receive real-time updates of project activities [13]; assignees may follow project owners and listen to their activities [14,15]. Watching and following activities show assignees' interests in projects, but do make real contributions for the project development.

GitHub offers two types of project owners, including personal account and organization. Personal account is intended for an individual developer, while organization is intended for a company or a non-profit organization, such as Google and Facebook.

In GitHub, any developers can participate in OSS projects by various ways, such as opening pull requests or submitting issues. However, only a set of developers are chosen by project administrators, and become assignees who are working on issues or pull requests. Personal accounts and organizations have different ways to choose assignees. Personal accounts usually manage a few projects, and they choose assignees for every project owned by them. Organizations often manage many more projects than personal accounts, and it is inconvenient to choose assignees for one project at a time. Organizations simplify management of many projects by teams[5]. Organizations build teams, and give teams special permissions to specific projects. Team members automatically become assignees of these projects. Teams allow organizations to add a group of developers as available assignees for several projects at once.

2.2. Research questions

We are interested in the following three research questions.

RQ1 Do assignees really participate in projects?

In GitHub, an issue or a pull request can be assigned to a specific assignee. No works have studied the activeness of available assignees. It remains unknown whether assignees are inactive or not. To the best of our knowledge, we take a first look at the existence of inactive yet available assignees. Based on 37 popular projects, we study the percentage of inactive assignees in projects, and compare projects with and without inactive assignees.

RQ2 Why do projects select inactive developers to become assignees?

In Section 4, we find that some projects have high percentage of inactive yet available assignees. GitHub allows projects to identify a set of developers who can be assigned to issues and pull requests. It is strange that some developers are chosen as assignees, but they never participate in projects. We study 17 projects with inactive assignees and 687 projects belonging to 8 organizations. We explore why some assignees are inactive but they become available assignees. We collect company information of inactive assignees from GitHub or LinkedIn, and then judge whether these inactive assignees work for organizations. We further discuss how organization management results in inactive yet available assignees.

RQ3 Do inactive assignees impact software development?

Developers' active participation is important for the sustainable development of OSS projects [16,17]. Assignees are responsible for working on some issues or pull requests. It remains unknown whether inactive assignees affect development of OSS projects. We randomly select some active developers in projects with inactive assignees. Then we send them questionnaires, and understand impacts of inactive assignees on software development.

3. Data collection

In this section, we firstly describe how we select projects for our research. Then we introduce how datasets are collected.

3.1. Project selection

We obtain 90 projects from MSR 2014 Mining Challenge dataset[6]. This dataset includes top-10 starred software projects for the top programming languages in GitHub. We focus on popular and active projects, because they may need assignees. In addition, small projects often have few assignees, and their project owners can manage projects by themselves.

Next, we collected basic information of 90 projects in January 2015. Then the following criteria is applied to select projects from the initial selection:

- Projects should have at least one event of activities within 1 month prior to data collection (January 2015), so as to avoid inactive projects.
- Projects should be created at least two years prior to data collection. It ensures that projects have more than two years of historical information. We are interested to explore how their assignees behave as time goes on.
- Projects should have at least 300 pull requests. In the pull-based software development, evaluating pull requests is an important task for assignees [18]. We choose projects which use pull requests to evaluate code contributions in GitHub.

[6] http://2014.msrconf.org/challenge.php.

After selection, our sample includes 37 projects. We introduce basic information of projects in Table 1. 13 projects were created earlier than December 2009, and have histories longer than 5 years; 29 projects have histories longer than 4 years, and 36 projects have histories longer than 3 years. Since GitHub was founded in April 2008[7], these projects have long histories in GitHub and provide great opportunities to explore their assignees. Our dataset includes projects written in representative programming languages, such as PHP, Ruby, Python, C, C++, JavaScript and Scala [13]. In GitHub, staring allows users to keep track of projects in which they are interested. The number of stars shows the popularity of the project. In our dataset, all projects have more than 1900 star, and 20 projects have more than 5000 stars. These results show that projects in our dataset are popular and active.

The third column in Table 1 describes types of project owners. GitHub offers two types of project owners, including personal account and organization. Personal account is intended for an individual developer, while organization is intended for a company or a non-profit organization, such as Google and Facebook. In our dataset, 27 projects owners are organizations, and 10 project owners are individual developers.

3.2. Data collection

GitHub provides access to its internal data through an API[8]. It allows us to access the rich collection of OSS projects, and provides valuable opportunities for research. We gather information through GitHub API and create datasets of projects and assignees.

According to the Section 2, we mainly collected activities in issues, pull requests and commits. We sent queries to GitHub API, received its replies, and extracted datasets since project creation time. For each issue, we crawled its ID, the developer who submitted it, the creation time and the close time. We also collected events related to this issue. Each issue had different types of events, and detailed description could be found in GitHub API[9]. The issue's comments were collected too, including submission time and commenters. For each pull request, we crawled its ID, the contributor, the creation time, the close time, related events and comments. For each commit, we collected its ID, the committer, the author, the commit time and comments. In GitHub, the author and the committer may be different. The author may be an ordinary developer, who submits modified codes through pull requests. The committer merges modified codes into the repository. Finally, we collected assignees through the issue assignee API [10]. This API returned all the available assignees to which issues or pull requests might be assigned.

We collected pull requests, issues, commits and assignees for 37 projects in January 2015. Table 2 shows basic statistics of projects. In total, we collected 2,374,474 records of activities and 1073 assignees for 37 projects. In the Section 4, we find projects *zipkin, paperclip* and *libuv* have high percentage of inactive assignees. These project owners are *twitter, thoughtbot* and *joyent*, which are large organizations. In Section 5.2, we explore whether other projects belonging to these owners also have high ratios of inactive assignees. Therefore, we collected datasets for 486 projects belonging to owners *twitter, thoughtbot* and *joyent* in April 2015. We further collected datasets for 201 projects belonging to owners *divio, zurb, plataformatec, libgit2* and *SignalR* in March 2017. Table 7 shows that we collected 797,756 records of activities for these 687 projects.

[7] https://github.com/blog/40-we-launched.
[8] http://developer.github.com/v3/.
[9] https://developer.github.com/v3/issues/events/.
[10] https://developer.github.com/v3/issues/assignees/.

Table 1
Basic information of projects.

Owner	Project	Owner Type	Language	Creation time	# Stars
twitter	zipkin	Org.	Scala	2012/6/6	1957
thoughtbot	paperclip	Org.	Ruby	2008/4/10	6246
joyent	libuv	Org.	C	2011/3/29	3055
divio	django-cms	Org.	Python	2009/3/5	2918
elasticsearch	elasticsearch	Org.	Java	2010/2/8	8067
zurb	foundation	Org.	CSS	2011/10/13	17,697
joyent	node	Org.	JavaScript	2009/5/27	29,852
plataformatec	devise	Org.	Ruby	2009/9/16	11,185
libgit2	libgit2	Org.	C	2010/9/10	4607
SignalR	SignalR	Org.	C#	2011/7/22	4403
reddit	reddit	Org.	Python	2008/6/18	6087
boto	boto	Org.	Python	2010/7/12	4036
openframe -works	openFrame -works	Org.	C	2009/10/21	2985
sbt	sbt	Org.	Scala	2009/8/17	1927
akka	akka	Org.	Scala	2009/2/16	2874
TrinityCore	TrinityCore	Org.	C++	2010/12/28	2536
xbmc	xbmc	Org.	C++	2011/1/3	3312
antirez	redis	User	C	2009/3/21	10,135
ariya	phantomjs	User	C++	2010/12/27	11,070
bitcoin	bitcoin	Org.	TypeScript	2010/12/19	5002
cakephp	cakephp	Org.	PHP	2010/5/8	5058
codeguy	Slim	User	PHP	2010/9/21	3904
diaspora	diaspora	Org.	Ruby	2010/9/15	9345
ginatrapani	ThinkUp	User	PHP	2009/6/6	2787
gitlabhq	gitlabhq	Org.	Ruby	2011/10/2	12,653
h5bp	html5 -boilerplate	Org.	JavaScript	2010/1/24	26,416
imathis	octopress	User	Ruby	2009/10/18	8216
kennethreitz	requests	User	Python	2011/2/13	10,759
mbostock	d3	User	JavaScript	2010/9/27	29,383
mitsuhiko	flask	User	Python	2010/4/6	10,591
mrdoob	three.js	User	JavaScript	2010/3/23	16,158
netty	netty	Org.	Java	2010/11/9	3402
rails	rails	Org.	Ruby	2008/4/11	22,969
scala	scala	Org.	Scala	2011/12/1	2800
sebastian -bergmann	phpunit	User	PHP	2009/12/24	3855
symfony	symfony	Org.	PHP	2010/1/4	8785
zendframework	zf2	Org.	PHP	2010/6/4	4829

4. Basic analysis of inactive assignees

In this section, we study activities of assignees, and analyze the percentage of inactive assignees in projects. Then we compare characteristics of projects with inactive assignees and projects without inactive assignees.

Based on datasets, we judge whether assignees ever participate in issues, pull requests or commits. For each project, we classify assignees into 2 kinds, namely active assignees and inactive assignees. Then we compute the percentage of inactive assignees, and plot results in Table 2. The project *zipkin* has the highest percentage of inactive assignees: Only 3.32% (9) of assignees actually participate in the project, while other 96.68% (262) of assignees never participate in the project. The percentage of inactive assignees is 66.35% in project *paperclip*, and the percentage of inactive assignees is 61.54% in project *libuv*. Issues or pull requests can be assigned to specific assignees. It is surprising that these projects have the percentage of inactive assignees higher than 60%. 7 projects have the percentage of inactive assignees between 20% and 40%, and 7 projects have the percentage of inactive assignees between 1% and 20%. It shows that inactive assignees are common in some projects.

Table 2 shows that inactive assignees exist in 17 projects, and 20 projects do not have any inactive assignees. We classify projects into 2 groups, including projects without inactive assignees, and projects with inactive assignees. In the following part, we compare characteristics of the 2 groups.

There are two types of project owners in GitHub, including personal owners and organizational owners. Personal account is intended for an individual developer, while organization account is intended for a company or a non-profit organization, such as Google and Facebook. In Table 3, we compare the owner type of projects with and without inactive assignees. For projects without inactive assignees, 10 projects belong to personal owners, and 10 projects belong to organizational owners. For projects with inactive assignees, 100% of projects belong to organizational owners. It shows that the owner type is an important factor of the activeness of assignees. In our datasets, all assignees are active in projects belonging to personal owners, and inactive assignees only appear in projects belonging to organizational owners.

The choice of programming languages reflects preference towards some kinds of tasks [13]. For example, PHP is applied in the development of web applications. We compare main programming languages used to implement projects with and without inactive assignees, and plot results in Table 4. Results show that projects with and without inactive assignees both use programming languages C, C++, Java, JavaScript, Python, Ruby and Scala. However, programming language PHP is used in 6 projects without inactive assignees, but it is used in 0 projects with inactive assignees. If a project uses programming language PHP, it is likely to be a project without inactive assignees.

Table 2
Statistics of assignees.

Project	# Activities	# Active developers	# Active assignees	# All assignees	# Inactive assignees
zipkin	3772	104	9	271	262 (96.68%)
paperclip	15,805	2384	35	104	69 (66.35%)
libuv	20,624	813	10	26	16 (61.54%)
django-cms	39,463	1130	19	42	23 (54.76%)
elasticsearch	111,411	3905	58	100	42 (42%)
foundation	56,693	5375	22	36	14 (38.89%)
node	112,962	5534	25	37	12 (32.43%)
devise	27,830	3462	15	21	6 (28.57%)
libgit2	41,336	615	29	39	10 (25.64%)
SignalR	35,872	1457	12	15	3 (20%)
reddit	13,037	882	15	18	3 (16.67%)
boto	26,133	1782	9	10	1 (10%)
openFrame -works	50,399	700	19	21	2 (9.52%)
sbt	21,738	834	11	12	1 (8.33%)
akka	89,885	491	13	14	1 (7.14%)
TrinityCore	166,534	2996	25	26	1 (3.85%)
xbmc	138,639	1328	73	75	2 (2.67%)
redis	23,692	1956	3	3	0 (0%)
phantomjs	23,642	3412	5	5	0 (0%)
bitcoin	86,950	1696	5	5	0 (0%)
cakephp	90,694	1084	19	19	0 (0%)
Slim	7778	601	2	2	0 (0%)
diaspora	74,445	1783	25	25	0 (0%)
ThinkUp	15,492	463	4	4	0 (0%)
gitlabhq	110,716	6134	13	13	0 (0%)
html5 -boilerplate	22,435	1736	6	6	0 (0%)
octopress	15,061	1603	3	3	0 (0%)
requests	34,267	1789	4	4	0 (0%)
d3	18,365	1267	2	2	0 (0%)
flask	11,834	1094	7	7	0 (0%)
three.js	65,711	3022	1	1	0 (0%)
netty	64,740	1046	18	18	0 (0%)
rails	320,155	11,867	45	45	0 (0%)
scala	97,970	463	18	18	0 (0%)
phpunit	18,771	1352	3	3	0 (0%)
symfony	196,419	4678	10	10	0 (0%)
zf2	103,204	1883	13	13	0 (0%)

Table 3
Comparison of owner type.

	Projects without inactive assignees	Projects with inactive assignees
Projects belonging to personal owners	10	0
Projects belonging to organizational owners	10	17

Table 4
Comparison of programming languages.

Programming language	Projects without inactive assignees	Projects with inactive assignees
C	1	3
C#	0	1
C++	1	2
CSS	0	1
Java	1	1
JavaScript	3	1
PHP	6	0
Python	2	3
Ruby	4	2
Scala	1	3
TypeScript	1	0

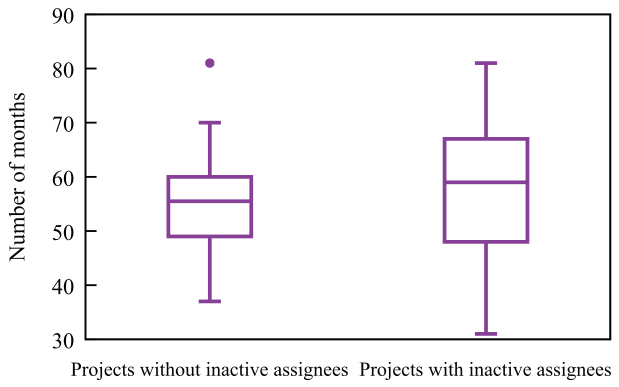

Fig. 1. Comparison of age.

We further explore whether projects with inactive assignees tend to be older or younger. The age of a project is defined as the number of months between its creation and the month of data collection. In Fig. 1, we plot the age distribution of projects with and without inactive assignees. The median value of the age of projects without inactive assignees is smaller than that of projects with inactive assignees. The Mann-Whitney-Wilcoxon (MWW) test is a non-parametric statistical test that assesses the statistical significance of the difference between two distributions [19]. Using the MWW test, we find that the difference between two groups of projects is not significant at 0.05 significance level. Therefore, the age cannot be used to distinguish between projects with and without inactive assignees.

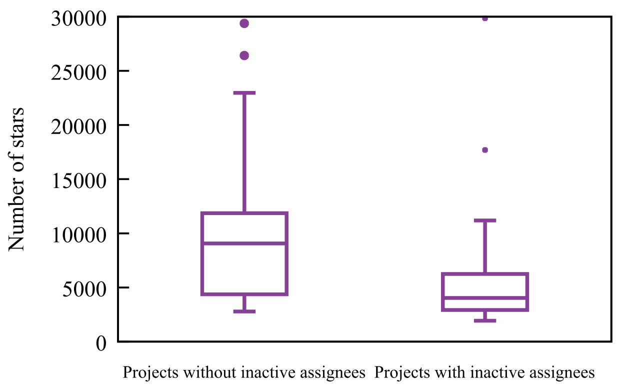

Fig. 2. Comparison of the number of stars.

Table 5
Feature importance.

Feature	Mean decrease accuracy (%)
Owner type	16.59
Programming language	9.82
Age	−1.51
Number of stars	5.56

In GitHub, staring allows users to keep track of projects in which they are interested. Fig. 2 plots the distribution of the number of stars for projects with and without inactive assignees. The median value of the star amount of projects without inactive assignees is higher than that of projects with inactive assignees. We also use the MWW test and get a p-value of 0.03548. The difference between two groups of projects is statistically significant at 0.05 significance level. This means that projects without inactive assignees have statistically significantly more stars than projects with inactive assignees. The number of stars shows the project popularity [20]. Projects without inactive assignees are more popular than projects with inactive assignees. In the survey of Section 6, we find that 37.25% of respondents describe that inactive assignees affect open source software development (i.e., causing unresolved issues or pull requests, and delaying software development). These negative impacts of inactive assignees may cause developers to lose interests in projects and decrease the project popularity. It is important to understand reasons of inactive assignees, and provide suggestions for projects to reduce inactive assignees and keep developers' interests.

Random forests are an ensemble learning method for regression, classification, and other tasks. Random forests construct a multitude of decision trees at training time and output the class that is mean prediction of trees. In order to further understand feature importance, we use random forests to compute mean decrease accuracy of features. Mean decrease accuracy reflects how much the average prediction accuracy decreases when randomly shuffled values are used for a particular feature in the testing phase [21]. A higher mean decrease accuracy value represents higher feature importance. Table 5 shows the mean decrease accuracy of features. Mean decrease accuracy is 16.59 for owner type. It means that using shuffled values for owner type affects the prediction accuracy by more than 16.59% relative to the performance with the correct values for this feature. Owner type is the most important feature in predicting whether a project has inactive assignees or not. Programming language and number of stars have mean decrease accuracies bigger than 0, and they are also useful to distinguish between projects with and without inactive assignees. Mean decrease accuracy is −1.51 for age, and age cannot be used to distinguish between projects with and without inactive assignees. Table 5 further proves results in Tables 3, 4, Figs. 1 and 2.

RQ1: Some projects have high percentage of inactive assignees.

5. Reasons for assignees being inactive

In this section, we firstly explore reasons for assignees being inactive. Table 2 shows that some projects have high percentage of inactive assignees. We study whether other projects belonging to the same owners also have high percentage of inactive assignees.

5.1. Company identification results

In GitHub, personal accounts are intended for individual developers. Personal owners promote developers as assignees for specific projects, and allow them to work on issues or pull requests. Organizations are used for creating distinct groups of users within companies, such as divisions or groups working on similar projects. Since organizations sometimes have many projects, they use teams to manage developers. Teams give organizations the ability to create groups of members and manage access permissions to projects. Once a team has been created, organization administrators give the team special permissions to specific projects. Then developers are added to the team, and have corresponding permissions to projects. Teams can map to physical teams within companies, but they can also represent areas of interest or expertise.

Table 3 also shows that inactive assignees only exist in projects belonging to organizational owners, and all assignees are active in projects belonging to personal owners. We doubt that inactive assignees may be caused by organizations. Some developers in companies may be added to organizations, and automatically become assignees of projects managed by some teams in organizations. However, these developers may never participate in some projects, and become inactive assignees.

In the following part, we explore whether inactive assignees work for organizations, and automatically become assignees. We obtain company names of organizations, collect company information of inactive assignees from GitHub or LinkedIn, and then judge whether these inactive assignees work for organizations.

In GitHub, the organization is intended for a company or a non-profit organization, such as Google and Facebook. Organization names are usually company names. For example, the organization *twitter* belongs to the company *Twitter*. However, there are some exceptions. For example, the organization *SignalR* belongs to the company *Microsoft*. We browse profiles of organizations in Table 1, and manually obtain their company names.

We collect company information of developers from GitHub. We send queries to GitHub API, and try to obtain company information of inactive assignees. Then we determine whether inactive assignees work in companies which projects belong to. For example, the developer *andypiper* works for the company *Twitter*. The project *zipkin* also belongs to the organization *twitter*, namely the company *Twitter*. Therefore, we decide that the developer *andypiper* becomes an inactive assignee of the project *zipkin*, because he works for the organization *twitter*. We obtain company information of 330 inactive assignees through GitHub API, and successfully confirm that 296 inactive assignees indeed work for the respective organizations which own the respective projects.

LinkedIn is a professional social networking site, where people build their professional identities online [22,23]. Though some developers do not fill in their companies in GitHub, they may write their companies in LinkedIn. In Table 2, we find 468 inactive assignees in 17 projects. We collect company information of 330 inactive assignees through GitHub API, and fail to obtain company information of other 138 inactive assignees. We try to find these

Table 6
Statistics of inactive assignees who are confirmed to work for the respective owner organizations.

Project	# Inactive assignees	Inactive assignees who are confirmed to work for the organization
zipkin	262	228 / 87.02%
paperclip	69	51 / 73.91%
libuv	16	14 / 87.5%
django-cms	23	12 / 52.17%
elasticsearch	42	27 / 64.29%
foundation	14	5 / 35.71%
node	12	9 / 75%
devise	6	6 / 100%
libgit2	10	9 / 90%
SignalR	3	3 / 100%
reddit	3	1 / 33.33%
boto	1	1 / 100%
openFrameworks	2	0 / 0%
sbt	1	1 / 100%
akka	1	1 / 100%
TrinityCore	1	1 / 100%
xbmc	2	0 / 0%

138 inactive assignees' company information from LinkedIn, and check whether they work in companies of projects.

In the advanced search of LinkedIn, we enter the name of an assignee and the company of the project. If the advanced search returns no results, we fail to find this inactive assignee in LinkedIn. If the advanced search returns any user, this user has the same name with the inactive assignee, and works in the company of the project in which this assignee is inactive. We further compare the returned user's photo in LinkedIn and the inactive assignee's photo in GitHub. If two profile photos are similar, we consider that the returned user in LinkedIn is the inactive assignee in GitHub, and the inactive assignee indeed works in the company of the project in which he or she is inactive; Otherwise, we fail to find this inactive assignee in LinkedIn. In total, we successfully confirm that 73 inactive assignees indeed work for organizations through LinkedIn.

We determine whether inactive assignees work for organizations through GitHub or LinkedIn, and plot results in Table 6. In project *zipkin*, 87.02% of inactive assignees are confirmed to work for the organization owning *zipkin*. In 11 projects, more than 70% of inactive assignees are confirmed to work for the respective organizations which own the projects.

Results show that the majority of inactive assignees indeed work for the respective owner organizations. Some organizations provide coarse-grained administration for teams. Organizations add workers in companies to teams, and these workers automatically become assignees. However, some workers may never participate in some projects.

In order to verify our finding, we randomly select 30 inactive assignees, and send them emails. We ask why they become inactive assignees, and receive 2 replies. An inactive yet available assignee says "I work at Twitter. We have 109 public repositories under our GitHub account. So I have access to all 109 repositories, but I am only active and handle pull requests in the ones that I work on (maybe 5 or 6 of them)." Another inactive assignee tells us that "I am listed as a collaborator of Zipkin because I am a member of the Twitter organization (my employer), and the Twitter organization is the owner of Zipkin. But I personally never work on Zipkin." Their replies further substantiate our findings.

5.2. Organization analysis

As shown in above subsections, inactive assignees are mainly caused by inappropriate administration of organizations owning the 17 projects. We take a further step, and explore whether other projects in the organizations also have inactive assignees.

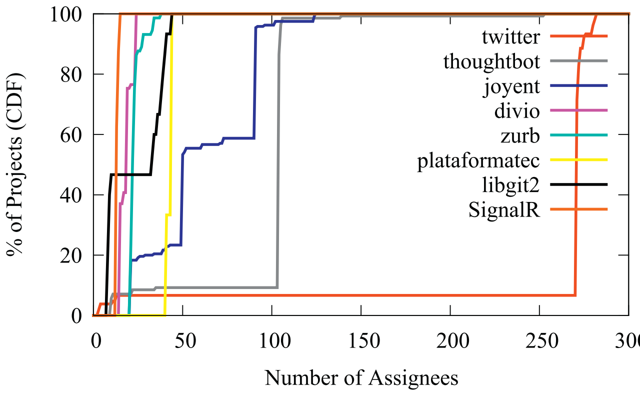

Fig. 3. Number of assignees in projects.

Table 2 shows that projects *zipkin, paperclip, libuv, django-cms, elasticsearch, foundation, node, devise, libgit2* and *SignalR* have percentages of inactive assignees higher than 20%. These 10 projects belong to 9 organizations. In March 2017, all projects sharing the same organization as *elasticsearch* are marked as private. We thus cannot collect details of these projects. The project*elasticsearch* was public in GitHub when we collected our first set of data in January 2015. However, its status changed and it was a private project on March 2017 when we collected our second set of data. We collected details of projects belonging to the following organizations: *twitter, thoughtbot, joyent, divio, zurb, plataformatec, libgit2* and *SignalR*. Table 7 shows statistic information of these projects. In total, we collect 797,756 records of activities for 687 projects belonging to the 8 organizations.

We compute the number of assignees in projects belonging to the 8 organizations, and plot cumulative distribution function (CDF) of number of assignees in Fig. 3. In the organization *twitter*, 6.67% of projects have less than 270 assignees, and the other 93.33% of projects have more than 270 assignees. In the organization *twitter*, the oldest project was created in April 2008, and the newest project was created in March 2015. Both old and new projects have many assignees, no matter how many assignees are really needed. For example, the project *unishark*[11] was created in March 2015, and had only 1 star when we collected our datasets. However, the project *unishark* has as many as 271 assignees. In the CDF, we can see a jump at 271. 68 (64.76%) projects all have 271 assignees. We guess that the organization *twitter* has a large team of 271 developers, and allows this team to manage many projects. 271 team members automatically become assignees of projects controlled by this team. 109 projects (77.3%) have 104 assignees in the organization *thoughtbot*. 88 (36.67%) projects have 91 assignees and another 72 (30%) projects have 50 assignees in the organization *joyent*. In Fig. 3, other organizations also have jump points. It shows that these organizations may also have some teams, whose members automatically become assignees of several projects. We cannot collect team information through GitHub API. In future, we will try to collect team information through other ways, and verify the existence of large teams in the organizations.

For projects belonging to the above 8 organizations, we identify inactive assignees who never participate in issues, pull requests or commits. Then we plot cumulative distribution function of the percentage of inactive assignees in Fig. 4. In the organization *twitter*, only 6.67% of projects have less than 90% assignees who are inactive, and the other 93.33% of projects have more than 90% inactive assignees. 70.47% of projects have even more than 97% inactive assignees. More than 80% of assignees are inactive in 84.4%, 95%,

[11] https://github.com/twitter/unishark.

Table 7
Basic information of projects in 8 organizations.

Organization		twitter	thoughtbot	joyent	divio
# Projects		105	141	240	81
# Activities		186,908	142,047	177,114	68,347
Common Language		Scala, Java JavaScript, Ruby	Ruby, Swift Objective-C, CSS	JavaScript, C Shell, Ruby	Python, JavaScript CSS, HTML
Creation time	Min	2008-4-21	2008-4-10	2009-4-27	2009-3-5
	Median	2012-5-7	2013-6-7	2014-8-27	2014-2-4
	Max	2015-3-12	2015-3-31	2015-3-27	2017-3-10
# Stars	Min	0	0	0	0
	Median	242	22	1	5
	Max	9,574	6,807	35,482	4765
Organization		zurb	plataformatec	libgit2	SignalR
# Projects		73	24	15	8
# Activities		80,962	35,914	63,926	42,538
Common Language		JavaScript, CSS HTML, Ruby	Ruby, Elixir JavaScript	C#, JavaScript C, Objective-C	Java, C# JavaScript
Creation time	Min	2010-10-13	2009-9-16	2010-9-10	2011-7-22
	Median	2014-10-10	2012-2-7	2011-2-27	2013-8-22
	Max	2017-1-30	2016-8-23	2015-3-10	2014-6-12
# Stars	Min	0	0	0	1
	Median	19	46	418	55
	Max	25,098	16,707	6,003	6,323

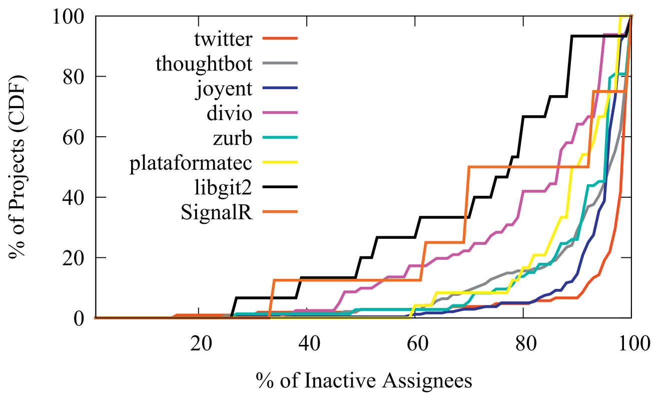

Fig. 4. Percentage of inactive assignees.

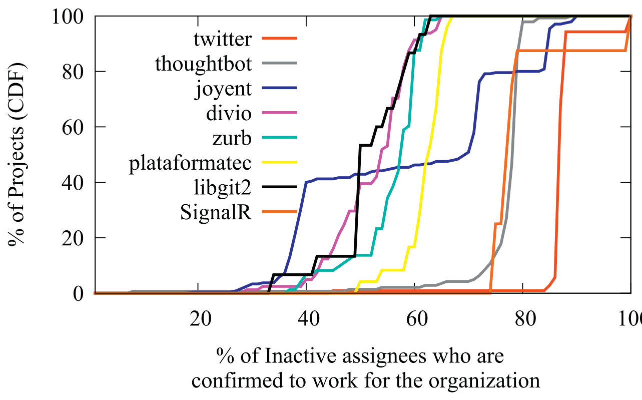

Fig. 5. Percentage of inactive assignees who are confirmed to work for the organization.

58.02%, 86.3% and 83.33% of projects in organizations *thoughtbot, joyent, divio, zurb* and *plataformatec*. More than 60% of assignees are inactive in 73.33% and 87.5% of projects in organizations *libgit2* and *SignalR*. In these organizations, most of their projects have high percentage of inactive assignees.

Finally, we collect additional information of inactive assignees through GitHub API or LinkedIn search, and investigate whether inactive assignees really work for the respective owner organizations. In Fig. 5, we plot the percentage of inactive assignees who are confirmed to work for the respective organizations which own the respective projects. In the organization *twitter*, only 2.86% of projects have less than 85% inactive assignees who are confirmed to work for *twitter*; for the other 97.14% of *twitter* projects, more than 85% of inactive assignees are confirmed to work for *twitter*. For 85.82% of projects in the organization *thoughtbot*, more than 75% of inactive assignees are confirmed to work for *thoughtbot*. For 52.5% of projects in the organization *joyent*, more than 65% of inactive assignees are confirmed to work for *joyent*. Other organizations have similar results. Results show that most inactive assignees indeed work for the respective organizations. When companies build large teams, many members in the teams may not participate in projects, but automatically become assignees.

RQ2: The main reason for developers being inactive assignees is that developers work for organizations and automatically become assignees of some projects in the organizations. However, these developers do not work on projects.

6. Impacts of inactive assignees

As shown in the Section 4, we find that some developers never participated in projects, but they are available assignees who can be assigned to issues or pull requests. We wonder whether these inactive yet available assignees were really assigned to issues or pull requests. As described in Section 3.2, we collected events of issues or pull requests. One event type is 'assignee', and it describes the developer who is assigned to an issue or a pull request. We analyze events of issues or pull requests, and identify inactive assignees who were really assigned to issues or pull requests in 17 projects. We find 6 projects with inactive assignees who were really assigned and list them in Table 8. For example, available assignee *simianhacker* was assigned to address an issue with num-

Table 8
Inactive assignees who were really assigned to issues or pull requests.

Project	Inactive assignees who were really assigned
django-cms	kkovrizhenko
elasticsearch	colinsurprenant, palecur, simianhacker
foundation	accbjt, jeaniec
SignalR	roopalik
openFrameworks	ofbot
xbmc	modhack

Table 9
Average number of days between assignment and close of issues or pull requests.

Project	Average number of days
django-cms	320
elasticsearch	213
foundation	23
SignalR	7
openFrameworks	60
xbmc	38

ber 7612 in project *elasticsearch* on September 7, 2014[12]. However, *simianhacker* never participated in the project, and he did not address this issue. The issue was finally closed on October 6, 2016, which was more than 2 year later after its submission. Another issue with number 7973 faced a similar problem[13]. Available assignee *palecur* was assigned to address this issue, but he was an inactive assignee. This issue was reassigned to another available assignee *dadoonet* 21 days later. The other 5 projects also have inactive assignees who were really assigned but never participated in the respective projects.

Six projects have inactive assignees who were really assigned to address issues or pull requests. For these issues or pull requests with inactive assignees, we compute the length of the time interval between assignment and close, and show them in Table 9. In project *elasticsearch*, issues or pull requests with inactive assignees have 213 days between assignment and close, which is a long time. In projects *django-cms, foundation, SignalR, openFrameworks* and *xbmc*, average number of days between assignment and close are 320, 23, 7, 60 and 38, respectively. Thus, issues or pull requests with wrong assignment of inactive assignees can cause serious delay.

We take a further step, and send questionnaires to active developers in projects to understand their attitudes towards inactive assignees. We ask 3 questions in the survey: (1) Before this survey, do you know that some assignees never participate in some projects? (2) In your opinion, do inactive assignees affect open source software development? (3) If you choose Yes in the question 2, what are impacts of inactive assignees on software development? In the first and the second questions, we provide predefined choices, including *yes, no* and *not sure*. The third question is open-ended, and allows respondents to describe impacts of inactive assignees in details.

As can be seen from Table 2, 17 projects have 33,792 active developers. In each project, we randomly choose 100 active developers, who fill in their email addresses in GitHub. Then we send emails to 1700 developers, and receive 102 replies. In the following paragraphs, we analyze the replies to understand active developers' attitudes towards inactive assignees.

[12] https://github.com/elastic/elasticsearch/issues/7612.
[13] https://github.com/elastic/elasticsearch/issues/7973.

Table 10
Before this survey, do developers know that some assignees never participate in some projects?

Response	Developers
Yes	53 / 51.96%
No	22 / 21.57%
Not sure	27 / 26.47%

Table 11
Do inactive assignees affect open source software development?

Response	Developers
Yes	38 / 37.25%
No	48 / 47.06%
Not sure	16 / 15.69%

Table 12
What are impacts of inactive assignees on software development?

Impact	Developers
Cause unresolved issues or pull requests	12 / 31.58%
Delay software development	11 / 28.95%
Make other contribution	5 / 13.16%
Other	5 / 13.16%
Not filled	5 / 13.16%

Firstly, we ask developers whether they know the existence of inactive assignees before the survey. We describe their answers in Table 10. 51.96% of respondents know that some assignees never participate in some projects before the survey. Though inactive assignees never appear in projects, the majority of respondents still notice the existence of inactive assignees.

Next, Table 11 shows developers' attitudes towards inactive assignees. 37.25% of respondents think that inactive assignees impact software development. 47.06% of respondents think that inactive assignees do not affect project development. Few respondents explain their reasons. A respondent says that "As long as the project still has engineers the tasks assigned to inactive assignees can just be reassigned." The other 15.69% of respondents are not sure about impacts of inactive assignees.

38 respondents think that inactive assignees bring impacts on software development. The first, third and fourth authors independently read their responses, and build corresponding categories. Three authors compare results, identify inconsistencies and retrofit categories. Then three authors classify responses independently. A response is classified into a specific category, if at least 2 authors make the same decision. If a response is classified into different categories, 3 authors discuss together, resolve conflicts and make decision. Table 12 shows impacts of inactive assignees on the software development. From the results, we can note that:

(1) One main impact is that inactive assignees may cause unresolved issues or pull requests. Inactive assignees give false assumption that issues or pull requests will be resolved, but nothing actually happen. These issues or pull requests get stalled and often never get finished. Active assignees will not work on issues or pull requests, if they think other people are already working on them. A respondent writes that "Off the top of my hat, if a person is assigned then others will be extremely unlikely to pick that work up, meaning that it may be abandoned forever". Another respondent says that "It impacts the workflow of a project. Inactive assignees who are assigned to issues will give a false sense of progress."

(2) 11 respondents think that inactive assignees cause delays in the open source software development. One example of their responses is "Inactive assignees might prevent some contributors

from fixing open tickets , thinking the fixes are being handled. It might cause delay in issues being handled." Another respondent writes "If the assignee is inactive that could delay a lot of things in a project and that could delay other downstream projects as well."

(3) 5 respondents think that inactive assignees may not participate in issues or pull requests, but they should contribute to OSS projects in other ways. For example, a respondent says that "Not all participants show up as activity on github. In my case that is providing know-how and guidance to active contributors and helping moderate communities." Another respondent writes that "Some people can improve the quality of the project just by correcting typo on documentation."

(4) 5 developers mention other impacts, and 5 developers do not fill in detailed impacts.

RQ3: 37.25% of respondents describe that inactive assignees affect open source software development (i.e., causing unresolved issues or pull requests, and delaying software development).

7. Discussion

7.1. Implications

Previous works have designed techniques to automatically recommended reviewers [24–27]. Some studies recommended reviewers for pull requests in GitHub [9–11]. Assignees are available developers to whom issues or pull requests may be assigned. Our findings suggest that some projects may have a lot of assignees, but only few assignees are active. If issues or pull requests are assigned to inactive assignees, these issues or pull requests will not be handled by these assignees and remain open for a long time, which may discourage submitters of issues or pull requests. Reviewer recommendation should exclude inactive yet available assignees for pull requests in GitHub. Future studies about assignees should be careful to perform data cleaning, since some available assignees are added by virtue of their employment and do not really work on projects.

These findings about inactive assignees imply that organizations should improve team management, and carefully select developers to become assignees in projects. In some organizations, workers may never participate in some projects, but become inactive assignees. Management confusion in some organizations causes high percentage of inactive assignees, which may confuse active developers in projects.

7.2. Threats to validity

Threats to internal validity relate to experimenter bias and errors. We collect datasets through GitHub API and decide whether assignees are active or not. The majority of inactive assignees are workers in companies. Some inactive assignees may discuss with other developers face to face at offices, and their contributions are not recorded in GitHub. Offline communications are not collected by us, which may affect the identification of inactive assignees.

Threats to external validity relates to the generalizability of our study. Our empirical findings are based on open source projects in GitHub, and it is unknown whether our results can be generalized to other OSS platforms, which also use teams to manage projects for large organizations. In the future, we plan to study a similar set of research questions by analyzing data from other platforms, and compare their results with our findings in GitHub.

Threats to conclusion validity is concerned with issues that affect the ability to draw the correct conclusion. Firstly, the most probable conclusion validity threat in our work is due to the analysis of questionnaires. We manually read replies from developers and analyze impacts of inactive assignees. Though these processes are subjective, our three authors read replies, compare results, identify inconsistencies and modify reasons, so as to reduce impacts from the experience and subjective awareness of authors. 17 projects have 33,792 active developers. 102 respondents from a population of 33,792 active developers yield a 95% confidence level with a 9.69% error margin. Secondly, we collect company information from GitHub or LinkedIn, and decide whether inactive assignees really work for the respective organizations which own the respective projects. Some inactive assignees may work for the respective owner organizations, but do not fill in their company information. Since we can not collect complete company information from GitHub or LinkedIn, actual proportions of inactive assignees working for the respective owner organizations may be larger than our results. It still shows that the majority of inactive assignees indeed work for the respective owner organizations.

Threats to construct validity are related to the degree which the construct being studied is affected by experiment settings. A frequently observed threat on a questionnaire-based analysis is that designed questions may misguide responders. We carefully design questions to ensure their understandability.

8. Related work

We firstly present related works about mechanisms in GitHub. Then we highlight related works on how developers make contributions and change roles in OSS projects.

8.1. Mechanisms in GitHub

In GitHub, an issue or a pull request can be assigned to a specific assignee who is responsible for working on the issue or pull request. Developers write issue reports to discuss bugs or feature requests. Developers open pull requests to submit code changes which they want to merge into original projects. Commits are used to record code changes in projects. Issues, pull requests and commits are important activities in GitHub [28]. As the popularity of GitHub, researchers study issues, pull requests or commits in GitHub [2], [3,4,29–40]].

Issue Bissyande et al. investigated the actual adoption of issue trackers in software development in GitHub [2]. They found that projects with reported issues tended to have more lines of codes and more developers. They further observed that the number of issues was strongly correlated with the number of watchers and forks.

Cabot et al. explored the use of labels to categorize issues [29]. They found that although the label mechanism was scarcely used, using labels favored the resolution of issues. They further reported that projects employed labels in different ways to classify issues according to priority, affected component and workflow process.

Pull request Gousios et al. performed an exploratory study of pull-based software development model in GitHub [3]. They observed that the decision to merge a pull request was mainly influenced by whether the pull request modified recently changed codes. They further found that the time to merge a pull request was influenced by the developer's previous track record, the project's size, its test coverage and its openness to external contributions.

Tsay et al. observed that both technical and social information were considered in the evaluation of pull requests [4]. They found that project managers made use of information signaling both good technical contribution practices for a pull request and the strength of the social relationship between the submitter and project manager in the evaluation of pull requests.

Jiang et al. proposed a method to predict whether integrators would be long-term active in the evaluation of pull requests [36]. They observed that whether integrators becoming long-term active was associated with the number of active months and social distance with contributors in their first year as integrators.

Commit Barnett et al. studied the relationship between commit message details and defect proneness in Java projects [39]. They found that their commit message volume metric and commit message content metric both contributed statistically significant amount of explanatory power to the Just-In-Time defect prediction models.

Michaud et al. designed an approach to automatically recover the previously unknown branch of origin of commits [40]. This approach used Git's default merge commit messages, and examined the relationships between neighboring commits. The evaluations showed that their approach had average accuracy higher than 97% and average precision higher than 80%.

Different from above works, we do not study detailed process of issues, pull requests or commits. We explore whether assignees really participate in issues, pull requests or commits. Then we study why some assignees are inactive, and analyze impacts of inactive assignees on OSS projects.

Organization Organization is intended for a company or a non-profit organization. We observe that some organization management causes inactive assignees. Few works studied organizations in GitHub. Gousios et al. developed a service to collaboratively collect and share data in GitHub [5]. He collected more than 900GB of raw data and 10GB of metadata, and allowed researchers to download GHTorent dataset. GHTorent dataset included members of organizations. This work provided valuable dataset for research, and it did not analyze organization management. This work is much different from our work.

8.2. Role migration

Developers have different roles in OSS projects. Some previous works studied developer contribution and their role migration in software development.

The structure of the hierarchical process was known as "onion model" [41,42]. They described the general layered structure of OSS communities, including project leaders, core members, active developers, peripheral developers, bug fixers, bug reporters, readers and passive users.

Zhou et al. observed that the probability for a new joiner to become a long term contributor was associated with her willingness and environment [17]. At the time of joining, future long term contributors tended to take more active role and showed more community-oriented attitude than other joiners. They also received more attention from the community and encountered more experienced peers.

Casalnuovo et al. found that developer migration in GitHub was strongly affected by pre-existing relationships [43]. They observed that joining a new project in which there were prior co-participants increased the developer's probability for initial contribution above baseline by 3.7% to 6.2%.

Above works mainly study factors which influence the role migration of active developers. However, these works do not consider inactive developers who have special roles in OSS projects. We study inactive assignees, which is different from above works.

9. Conclusion

This paper presents an empirical study of inactive yet available assignees in GitHub. We show empirical evidence that: (1) Some projects have high percentage of inactive assignees. Projects with inactive assignees all belong to organizational owners, rather than personal owners. (2) The main reason for developers being inactive assignees is that developers work for organizations and automatically become assignees of some projects in the organizations. However, these developers do not work on projects. (3) 37.25% of respondents describe that inactive assignees affect open source software development (i.e., causing unresolved issues or pull requests, and delaying software development).

Our findings provide implications for organizations and future sturdies. Organizations should carefully manage team members, and select suitable developers to become assignees. Currently, some organizations provide coarse-grained administration for teams. Organizations add workers in companies to teams, and these workers automatically become assignees. However, some workers may never participate in some projects. It causes the high percentage of assignees. Assignees are available developers to whom issues or pull requests may be assigned. If issues or pull requests are assigned to inactive assignees, these issues or pull requests will not be handled by these assignees and remain open for a long time, which may discourage submitters of issues or pull requests. Reviewer recommendation should exclude inactive yet available assignees for pull requests in GitHub. Future studies about assignees should be careful to perform data cleaning, since some available assignees are added by virtue of their employment and do not really work on projects.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant no. 61300006, and the State Key Laboratory of Software Development Environment under Grant no. SKLSDE-2017ZX-06, and Beijing Natural Science Foundation under Grant no. 4163074.

References

[1] A. Lima, L. Rossi, M. Musolesi, Coding together at scale: Github as a collaborative social network, in: Proc. of AAAI, Qubec, Canada, 2014.
[2] T.F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, Y.L. Traon, Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub, in: Proc. of ISSRE, Washington DC, USA, 2013.
[3] G. Gousios, M. Pinzger, A. van Deursen, An exploratory study of the pull-based software development model, in: Proc. of ICSE, Hyderabad, India, 2014.
[4] J. Tsay, L. Dabbish, J. Herbsleb, Influence of social and technical factors for evaluating contribution in GitHub, in: Proc. of ICSE, Hyderabad, India, 2014.
[5] G. Gousios, The ghtorent dataset and tool suite, in: Proc. of MSR, San Francisco, USA, 2013.
[6] M. Foucault, M. Palyart, X. Blanc, G.C. Murphy, J.-R. Falleri, Developer turnover in open-source software, in: Proc. of FSE, Bergamo, Italy, 2015.
[7] K. Crowston, K. Wei, J. Howison, A. Wiggins, Free/libre open source software development: what we know and what we do not know, ACM Comput. Surv. 44 (2012).
[8] J. Xavier, A. Macedo, M. de Almeida Maia, Understanding the popularity of reporters and assignees in the GitHub, in: Proc. of SEKE, Vancouver, Canada, 2014.
[9] Y. Yu, H. Wang, G. Yin, C. Ling, Reviewer recommender of pull-requests in GitHub, in: Proc. the 30th ICSME, Victoria, Canada, 2014a, pp. 609–612.
[10] Y. Yu, H. Wang, G. Yin, C.X. Ling, Who should review this pull-request: reviewer recommendation to expedite crowd collaboration, in: Proc. the 21st APSEC, Jeju, Korea, 2014b, pp. 335–342.
[11] J. Jiang, J.-H. He, X.-Y. Chen, Coredevrec: automatic core member recommendation for contribution evaluation, J. Comput. Sci. Technol. 30 (5) (2015) 998–1016.
[12] K. Herzig, S. Just, A. Zeller, Its not a bug, its a feature: how misclassification impacts bug prediction, in: Proc. of ICSE, San Francisco, USA, 2013.
[13] T.F. Bissyande, F. Thung, D. Lo, L. Jiang, L. Reveillere, Popularity, interoperability, and impact of programming languages in 100,000 open source projects, in: Proc. of COMPSAC, Kyoto, Japan, 2013.
[14] J. Jiang, L. Zhang, L. Li, Understanding project dissemination on a social coding site, in: Proc. of WCRE, Koblenz, Germany, 2013.
[15] L. Dabbish, C. Stuart, J. Herbsleb, Social coding in GitHub: transparency and collaboration in an open software repository, in: Proc. of CSCW, Washington, USA, 2012.
[16] M. Zhou, A. Mockus, Does the initial environment impact the future of developers? in: Proc. of ICSE, Honolulu, USA, 2011.
[17] M. Zhou, A. Mockus, What make long term contributors:willingness and opportunity in OSS community, in: Proc. of ICSE, Zurich, Switzerland, 2012.

[18] G. Gousios, A. Zaidman, M.-A. Storey, A. van Deursen, Work practices and challenges in pull-based development: the integrators perspective, in: Proc. of ICSE, Florence, Italy, 2015.
[19] H.B. Mann, D.R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Ann. Math. Stat. 18 (1) (1947) 50–60.
[20] H. Borges, A. Hora, M.T. Valente, Predicting the popularity of GitHub repositories, in: Proc. of PROMISE, Ciudad Real, Spain, 2016.
[21] S. Kuhn, B. Egert, S. Neumann, C. Steinbeck, Building blocks for automated elucidation of metabolites: machine learning methods for NMR prediction, BMC Bioinf. 9 (2008) 1–19.
[22] D. Kim, J.-H. Kim, Y. Nam, How does industry use social networking sites? An analysis of corporate dialogic uses of facebook, twitter, youtube, and linkedin by industry type, Qual. Quant. 48 (2014) 2605–2614.
[23] A. Archambault, J. Grudin, A longitudinal study of facebook, linkedin, and twitter use, in: Proc. of CHI, Austin, USA, 2012.
[24] J.B. Lee, A. Ihara, A. Monden, K. ichi Matsumoto, Patch reviewer recommendation in OSS projects, in: Proc. of APSEC, Bangkok, Thailand, 2013.
[25] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: Proc. the 35th ICSE, San Francisco, USA, 2013, pp. 931–940.
[26] P. Thongtanunam, C. Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K. ichi Matsumoto, Who should review my code? A file location-based code-reviewer recommendation approach for modern code review, in: Proc. the 22nd SANER, Montreal, Canada, 2015, pp. 141–150.
[27] X. Xia, D. Lo, X. Wang, X. Yang, Who should review this change? in: Proc. of ICSME, Bremen, Germany, 2015.
[28] C. Bird, P.C. Rigby, E.T. Barr, D.J. Hamilton, D.M. German, P. Devanbu, The promises and perils of mining git, in: Proc. of MSR, Vancouver, Canada, 2009.
[29] J. Cabot, J.L.C. Izquierdo, V. Cosentino, B. Rolandi, Exploring the use of labels to categorize issues in open-source software projects, in: Proc. of SANER, Montreal, Canada, 2015.
[30] R. Kikas, M. Dumas, D. Pfahl, Issue dynamics in GitHub projects, PROFES, Bolzano, Italy, 2015.
[31] V.J. Hellendoorn, P.T. Devanbu, A. Bacchelli, Will they like this? Evaluating code contributions with language models, in: Proc. of MSR, Florence, Italy, 2015.
[32] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, V. Filkov, Quality and productivity outcomes relating to continuous integration in GitHub, in: Proc. of FSE, Bergamo, Italy, 2015.
[33] Y. Yu, H. Wang, V. Filkov, P. Devanbu, B. Vasilescu, Wait for it: determinants of pull request evaluation latency on GitHub, in: Proc. of MSR, Florence, Italy, 2015.
[34] E. van der Veen, G. Gousios, A. Zaidman, Automatically prioritizing pull requests, in: Proc. of MSR, Florence, Italy, 2015.
[35] J. Tsay, L. Dabbish, J. Herbsleb, Let's talk about it: evaluating contributions through discussion in GitHub, in: Proc. of ICSE, Florence, Italy, 2015.
[36] J. Jiang, F. Feng, X. Lian, L. Zhang, Long-term active integrator prediction in the evaluation of code contributions, in: Proc. of SEKE, San francisco, USA, 2016.
[37] E. Guzman, D. Azcar, Y. Li, Sentiment analysis of commit comments in GitHub: an empirical study, in: Proc. of MSR, Hyderabad, India, 2014.
[38] V. Sinha, A. Lazar, B. Sharif, Analyzing developer sentiment in commit logs, in: Proc. of MSR, Austin, USA, 2016.
[39] J.G. Barnett, C.K. Gathuru, L.S. Soldano, S. McIntosh, The relationship between commit message detail and defect proneness in java projects on GitHub, in: Proc. of MSR, Austin, USA, 2016.
[40] H. Michaud, D. Guarnera, M. Collard, J. Maletic, Recovering commit branch of origin from GitHub repositories, in: Proc. of ICSME, Raleigh, USA, 2016.
[41] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, Y. Ye, Evolution patterns of open-source software systems and communities, International Workshop on Principles of Software Evolution, Orlando, USA, 2002.
[42] Y. Ye, K. Kishida, Toward an understanding of the motivation open source software developers, in: Proc. of ICSE, Portland, USA, 2003.
[43] C. Casalnuovo, B. Vasilescu, P. Devanbu, V. Filkov, Developer onboarding in GitHub: the role of prior social links and language experience, in: Proc. of FSE, Bergamo, Italy, 2015.