Empir Software Eng
DOI 10.1007/s10664-016-9436-6

Why and how developers fork what from whom in GitHub

Jing Jiang[1] · David Lo[2] · Jiahuan He[1] · Xin Xia[3] · Pavneet Singh Kochhar[2] · Li Zhang[1]

Abstract Forking is the creation of a new software repository by copying another repository. Though forking is controversial in traditional open source software (OSS) community, it is encouraged and is a built-in feature in GitHub. Developers freely fork repositories, use codes as their own and make changes. A deep understanding of repository forking can provide important insights for OSS community and GitHub. In this paper, we explore *why* and *how* developers fork *what* from *whom* in GitHub. We collect a dataset containing 236,344 developers and 1,841,324 forks. We make surveys, and analyze programming languages and owners of forked repositories. Our main observations are: (1) Developers fork repositories to submit pull requests, fix bugs, add new features and keep copies etc. Developers find repositories to fork from various sources: search engines, external sites (e.g., Twitter, Reddit), social relationships, etc. More than 42 % of developers that we have surveyed agree that an automated recommendation tool is useful to help them pick repositories to fork, while more than 44.4 % of developers do not value a recommendation tool. Developers care about repository owners when they fork repositories. (2) A repository written in a developer's preferred programming language is more likely to be forked. (3) Developers mostly fork repositories from creators. In comparison with unattractive repository owners, attractive repository owners have higher percentage of organizations, more followers and

Communicated by: Massimiliano Di Penta

Li Zhang is the first corresponding author

✉ Jing Jiang
jiangjing@buaa.edu.cn

✉ Li Zhang
lily@buaa.edu.cn

[1] State Key Laboratory of Software Development Environment, Beihang University, Beijing, China

[2] School of Information Systems, Singapore Management University, Singapore, Singapore

[3] College of Computer Science and Technology, Zhejiang University, Hangzhou, China

Published online: 31 May 2016

earlier registration in GitHub. Our results show that forking is mainly used for making contributions of original repositories, and it is beneficial for OSS community. Moreover, our results show the value of recommendation and provide important insights for GitHub to recommend repositories.

Keywords Fork · Open source software · GitHub

1 Introduction

Forking is the creation of a new software repository by copying another repository, without the permission from repository owner (Ernst et al. 2010). The free software community has different attitudes towards forking. On the negative side, forking is considered as a danger to OSS development (Nagy et al. 2010). This is the case since forking is regarded as introducing duplication of effort, reducing communication and splitting up of a repository into competitive and incompatible versions (Muffatto and Faldani 2003). On the positive side, the beauty of OSS development is that no permission from the authors is needed to start a fork. Moreover, forking satisfies different kinds of needs, such as adding new features, fixing bugs and meeting commercial strategy (Robles and Gonzalez-Barahona 2012).

Though forking is controversial in traditional OSS community, GitHub[1] encourages frequent forking as a built-in feature of its infrastructure. GitHub is a web-based hosting service for software development repositories and one of the largest and most popular open source communities in the world. In GitHub, developers are allowed to fork repositories and make changes without asking for permission. Developers can submit pull requests when they want to merge their changes into the repositories they fork from. This "fork and pull" model separates making modification and integrating change, and makes contributing to others' repositories much easier than it has ever been (Kalliamvakou et al. 2014). A recent study finds that 14 % of the repositories in GitHub adopt the "fork and pull" model (Gousios et al. 2014).

A deep understanding of code forking in GitHub can provide important insights for the OSS community and GitHub: Firstly, forking is allowed and fully supported in GitHub. It remains unknown whether forking is good or bad for OSS projects. Forking is the precursor to contributing back to the parent repository, and the convenience of the 'fork and pull' model may attract contributions from developers. However, forking allows developers to freely modify repositories, and even generates competitive and incompatible versions of the original repositories. For example, the repository *rowanj/gitx*[2] is a parallel variant that makes improvements over the original repository *pieter/gitx*.[3] The effect of forking mainly depends on the reasons why developers fork repositories. It is important to explore whether developers fork repositories for making contributions or building other competitive projects. Secondly, current recommendation in GitHub includes trending repositories, or repositories starred by friends. However, developers fork different repositories based on their interests. The understanding of forking can guide GitHub to improve its operation, for example personalized recommendation of repositories which developers are likely to fork.

[1] http://github.com

[2] https://github.com/rowanj/gitx

[3] https://github.com/pieter/gitx

Despite the above-mentioned benefits, there have been limited studies that investigate how developers fork repositories in GitHub. Thus, there is a need for a comprehensive analysis of forking behavior based on a large number of repositories. In this paper, we fill this need by investigating 236,344 developers and 1,841,324 forks, and analyze the characteristics of forking behavior. In particular, our study aims to answer four key questions:

1. Why developers fork repositories? How developers find these repositories? Do developers value an automatic recommendation system to fork repositories? Do developers care about repository owners?
2. What kinds of repositories do developers fork? Do developers prefer to fork repositories written in a particular programming language?
3. From whom do developers fork repositories? What are some characteristics of attractive owners of repositories that get forked?

To answer these essential and practical questions, we analyze repository forking from different perspectives. Our study provides a number of insights into forking behavior in GitHub.

– Developers fork repositories to submit pull requests, fix bugs, add new features and keep copies etc. Developers find repositories to fork by using various search engines, by reading content on external sites (e.g., Twitter, Hacker News, Reddit, etc.), by social relationships (i.e., forking repositories forked by developers that they follow in GitHub), by noting GitHub recommendation (e.g., trending repositories) on its explore page, etc. More than 42 % of developers that we have surveyed agree that an automated recommendation tool is useful to help them pick repositories to fork, while more than 44.4 % of developers do not value a recommendation tool. 35.2 % of respondents care about repository owners when they fork repositories.
– Developers are likely to fork repositories written in their preferred programming languages.
– Developers mostly fork repositories from creators. In comparison with unattractive repository owners, attractive repository owners have higher percentage of organizations, more followers and earlier registration in GitHub.

Our results provide important insights for the OSS community and GitHub. Results show that forking does not split developers into different competing and incompatible versions of repositories. Instead, the main cause of forking is submitting pull requests and making contributions to the original repositories. Moreover, most developers directly fork repositories from original creators, and contribute to these original repositories (rather than repositories forked by others). Therefore, forking is useful to attract contributions and it is beneficial for the OSS community. Currently, GitHub recommends trending repositories, or repositories stared by friends. Our results provide new insights for GitHub to make personalized recommendation. More specifically, GitHub can recommend repositories written in developers' preferred programming languages or created by older organizations with more followers.

2 Data Collection and Initial Analysis

Before diving into detailed analysis of repository forking, we begin by providing background information about GitHub. Then we introduce how our datasets are collected and present our initial analysis.

2.1 Forking in GitHub

GitHub uses Git (Bird et al. 2009) as its distributed revision control and source code management system. GitHub had over 10.6 million repositories as of January 2014 (Kalliamvakou et al. 2014). It is one of the largest and most popular open source communities in the world.

GitHub uses "fork & pull" model for collaboration on distributed software development. Developers fork repositories, use code as their own and make changes that they may contribute back through pull requests. The possibility of forking ensures that any developer has the opportunity to freely modify codes and make own decisions within the limits of their willingness and capacity to contribute. Forking is the precursor to contributing back to the parent repository. Developers can submit some code changes to the parent repository by issuing pull requests (Gousios et al. 2014). The parent repository's owner evaluates potential contributions of code changes, and decides whether to accept requests and merges these changes back into the repository. By separating the concerns of building projects and integrating changes, work is cleanly distributed between a contributor team that submits changes to be considered for merging and a core team that oversees the merging process, provides feedback, conducts tests, requests changes, and finally accepts the contributions (Kalliamvakou et al. 2014).

By allowing people to fork a repository and make changes without asking for permission, GitHub has made contributing to others' repositories much easier than it has ever been. The forking mechanism provides good accessibility of codes and attracts many eager helpers who like to make contributions. As more developers fork the repository, more developers are familiar with the code base, and they are able to evolve the software and satisfy different requirements.

2.2 Data Collection

GitHub provides access to its internal data stores through an API.[4] It allows researchers to access a rich collection of information about developers and repositories, and provides valuable opportunities to understand forking behavior.

Crawling personal information. In our previous work (Jiang et al. 2013), we proceeded to perform a breadth-first traversal of social graphs through GitHub API. In total, we collected 747,107 developers and their 2,234,845 social links. We took a further step and crawled personal information of 747,107 accounts in October 2013. For each account, we collected email address, type, registration date, location, company and blog link. GitHub offers two types of accounts, namely developers and organizations. Details of account types are described in Section 5.2. Personal information is valuable to thoroughly understand developers.

Collecting repositories. In GitHub, developers create repositories by themselves, or fork others' repositories. No matter whether repositories are created or forked, developers can freely modify code and consider these repositories as their own. We accessed GitHub's API and collected 4,344,316 repositories of 747,107 developers in October, 2013. 2,090,423 repositories are forked, and 2,253,893 repositories are created. Each piece of information includes the identifier of the repository, the identifier of the developer, the creation time, the updated time, the programming language and the create-fork flag. A repository may have

[4] http://developer.github.com/v3/

code written in various languages (FBissyande et al. 2013), and the major programming language is recorded. The create-fork flag indicates whether a repository is created or forked. For a forked repository, the creation time is the time when the developer forks the repository.

If a repository is forked, it has a *parent repository* and a *source repository*: The developer forks the parent repository and copies its code to create the repository. The parent repository may also be forked from another repository. If this is the case, the parent repository and its parent and ancestors form a fork tree. The source repository is the root of this fork tree. Different from the parent repository, the source repository is created rather than forked from another repository. For example, repository A is created and it is not forked from others. Repository B is forked from repository A, and repository C is forked from repository B. For repository C, repository B is the parent repository and repository A is the source repository. For forked repositories, we also collected detailed information of their parent repositories and source repositories. These datasets are valuable to analyze repository forking.

2.3 General Statistics

We compute some basic statistics to understand developers' attitude towards repository forking in GitHub. We begin by analyzing the distribution of forked repositories across the developer population. For each developer, we compute the number of forked repositories. We then aggregate across all developers the amount of forked repositories, and plot cumulative distribution functions (CDFs) in Fig. 1. In the line 'Total Developers', a point (x, y) in the line means that y % of developers have less than or equal to x forked repositories. In the line 'Total Forked Repositories', the point (x, y) means that y % of forked repositories belong to developers who have less than or equal to x forked repositories. Figure 1 shows that 68.4 % of developers have less than or equal to 4 forked repositories, but these developers only have 15.2 % of forked repositories. The other 31.6 % of developers have more than 4 forked repositories, and these developers have 84.8 % of forked repositories. We observe that the majority of developers fork few repositories, and they only have the minority of forked repositories. In this paper, we focus on developers who are active in forking repositories. Therefore, we exclude developers with less than 5 forked repositories, and 236,344 developers are left in our datasets. In total, we analyze these 236,344 developers and their 1,841,324 forked repositories. Though 236,344 developers only cover 31.6 % of all developers, they have 84.8 % of forked repositories.

Next, we wonder whether developers have repositories forked from others more than repositories created by their own. For each developer, we compute the number of his/her forked repositories, divided by the number of all repositories created or forked by him/her. Figure 2 shows a cumulative distribution function (CDF) of the percentage of forked repositories. The distribution is quite even: 38.3 % of developers have less than 30 % of

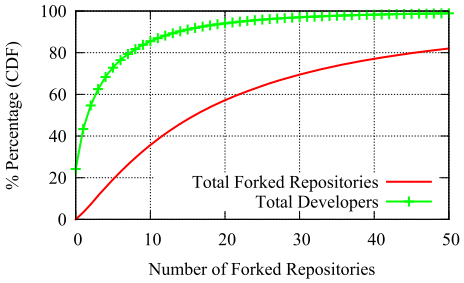

Fig. 1 Distribution of the number of forked repositories

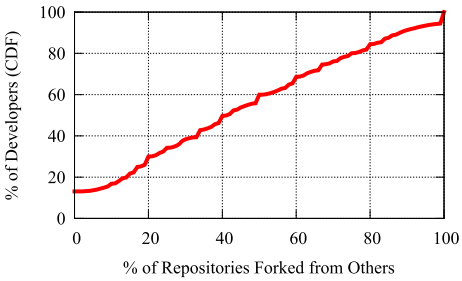

Fig. 2 Percentage of forked repositories

repositories forked from others; 60 % of developers have less than 50 % of repositories forked from others, which means that 40 % of developers have more than 50 % of repositories forked from others. For 40 % of developers, the number of forked repositories is more than the number of repositories created by their own.

We take a further step and study developers' attitude towards forking over time. GitHub was founded in April 2008.[5] For each month since GitHub's was founded, we compute the number of forked repositories divided by the number of all repositories created or forked in that month. Figure 3 shows the percentage of forked repositories as time evolves. In general, the line increases steadily and developers are more willing to fork repositories over time. Since month 49 (May 2012), the percentage of forked repositories has become greater than 50 %, and developers fork repositories more than they create repositories. As time evolves, developers become more active in forking repositories.

3 Why and How Do Developers Find Repositories to Fork?

Before a developer forks a repository, the developer needs to decide the repository that he/she is interested in. There are more than ten million repositories in GitHub and thus it takes some effort to find a repository of interest. To understand why and how developers find repositories that they fork, we designed questionnaires and run 2 rounds of survey. In the first round, we sent developers emails, and asked open-ended questions. After analyzing results of the first round, we identified potential answers and designed multiple choice questions. We also added some related and open-ended questions to make deep investigation. We sent developers emails, and asked new designed questions. In this section, we firstly describe our research method in details, and then we present the findings of our investigation.

3.1 Survey Design and Execution

We performed two rounds of survey to get a better insight into developer perception. In the first round, we only asked a few questions, to get a few insights. These insights led to more unanswered questions, which motivated us to conduct a second round of survey. With the second survey round, we can also confirm or reject findings that are found in the first round.

In the first round, we ran a pilot survey with three questions. These questions were: 1) Why do developers fork repositories? 2) How do developers find repositories that they fork later? 3) Will it be useful or useless if there is a tool to recommend repositories which

[5] https://github.com/blog/40-we-launched

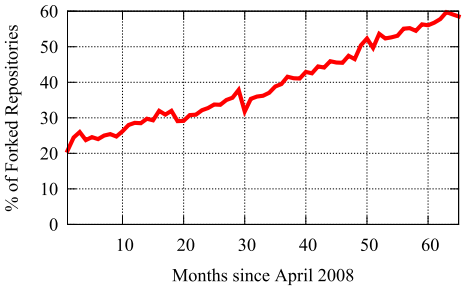

Fig. 3 Percentage of forked repositories over time

developers are likely to fork? We randomly selected 1,000 developers who had forked at least 30 repositories, sent them emails and asked open-ended questions from January 26 to February 23, 2014. In total, we received 124 replies.

Since the first two questions were open-ended. Two authors manually analyzed responses and built codes. The two authors compared results and agreed on the final set of codes. These codes were automatically applied on all replies to assign categories to each of the replies.

In the second round, we asked 7 questions instead of 3. The 7 questions were: 1) Why do developers fork repositories? 2) If creating pull requests is one of main reasons, what are purposes of pull requests? 3) How do developers find repositories that they fork later? 4) When developers fork repositories, do they care about who repository owners are? Why? 5) Will it be useful or useless if there is a tool to recommend repositories which developers are likely to fork? 6) If the recommendation tool is useful, what features of recommendation tool would developers like to have? 7) If the recommendation tool is useless, why?

We included four additional questions (questions 2, 4, 6 and 7) to investigate the purpose of developers performing pull requests, whether developers care about owners of repositories they fork from, features of a recommendation tool that developers like, and why some developers think that a recommendation tool is not useful. These were questions that arised after the responses to the first survey round were analyzed. For example, replies in the first round show that submitting pull requests is the most common reason for forking repositories. In the second round, we added a question to understand purposes of pull requests. Also, in the first round, we only asked developers' attitudes towards the recommendation tool. In the second round, we added two questions to make further exploration. If a developer considered the recommendation tool as useful, we asked the developer to provide desired features of a recommendation tool. If a developer considered the recommendation tool as useless, we asked for reasons.

Three out of the seven questions (questions 1, 3 and 5) also appeared in the first round. We included them so that we can confirm or reject findings of the first survey round. We changed questions 1 and 3 to be multiple choices questions instead of open-ended questions. We analyzed results of the first round, and provided predefined answers for questions 1 and 3. We did this step so that we could map answers that participants provided to these two questions across the two survey rounds. Still, to not limit developer responses and further elicit developer opinions, we allowed the respondents to choose "other" as a response, and allowed respondents to write other answers.

In the second round, we randomly selected 3,000 developers who have forked at least 30 repositories, sent them emails and asked them to answer the 7 questions from April 29 to May 8, 2015. In total, we received 162 replies.

In the following paragraphs, we first present root causes of forking behavior. Then, we analyze mechanisms through which developers find repositories and fork them. Thirdly, we study their preference towards repository recommendation. Finally, we explore the importance of repository owners.

3.2 Root Causes

To examine why developers fork some repositories, we email 1,000 developers in the first-round survey. We receive 124 replies which we qualitatively analyze. We use open coding to come up with a set of reasons of why developers fork these repositories as follows: The first author reads all the replies and understands the reasons that developers provide for them to fork repositories. Based on this understanding, the first author build codes for the reasons why developers fork repositories. For example, "fixing bugs" is often mentioned in the replies, and the corresponding code is "fix bug". This process is repeated by the third author who comes up with another set of codes. Finally, the two authors compare their results and agree on the final set of codes and reasons.

At the end of the above-mentioned process, we divide the reasons developers fork repositories into 4 categories. These categories and their representative codes are shown in Table 1. The codes are applied on all the replies to assign one or more of the 4 categories to each of the replies. If a reply includes all the codes in a set, the corresponding reason will be assigned to it. Note that a developer may mention several reasons, and thus multiple categories can be assigned to a reply. The second column in Table 1 shows the number of developers that mention different reasons for forking. Since multiple categories can be assigned to a reply, the percentages in Table 1 do not sum up to 100 %. From the results we can note that:

1) Submitting pull requests is the most common reason, which covers 46 % of replies. Forking is the precursor to contributing back to the parent repositories in GitHub (Gousios et al. 2014). Developers fork repositories, make modifications and submit pull requests back to original repositories. 46 % of repliers fork repositories, because they want to make contributions to the original repositories, rather than generating some competitive and incompatible versions. Forking does not split developers for different versions of repositories. Submitting pull requests is sometimes mentioned after fixing bugs or adding features. This is because sending pull requests is the main mechanism for submitting modification back to the original repositories in GitHub.
2) Fixing bugs is the second common reason and it is mentioned by 36.3 % of respondents. A developer says that "When I depend on a project and it has a bug, I fix the bug, then

Table 1 Root causes of repository forking

Cause	Developers in the first round	Developers in the second round	Code set
Submit pull requests	57 / 46 %	128 / 79 %	Submit pull request, contribute
Fix bugs	45 / 36.3 %	125 / 77.2 %	Fix bug, bugfix
Add features	27 / 21.8 %	112 / 69.1 %	Add feature
Keep copies	11 / 8.9 %	81 / 50 %	Copy
Other	N/A	8 / 4.9 %	N/A

I fork the project and add the bug fix." Some developers fork repositories, fix bugs and contribute modification back to parent repositories.

3) 27 respondents fork repositories and add new features to extend project functionalities. Some developers want to implement features they need in repositories. Therefore, they fork repositories and add new features.
4) 11 respondents decide to fork repositories because they want to keep copies in case the parent repository owners delete their repositories in GitHub. It ensures that when developers need to tweak a code from a repository in GitHub to suit their own needs, they can always do so and their work is tracked and backed up on GitHub, whatever the parent repository owners choose to do.

In the first round, the question was open-ended, and developers described reasons. According to results in the first round, we designed predefined answers and provided multiple choices in the second round. Then we sent emails to 3,000 developers, and received 162 replies. The third column in Table 1 shows the number of developers who choose reasons for forking in the second round. Results of the second round confirm results of the first round. Main reasons of forking include submitting pull requests, fixing bugs, adding features and keeping copies. Only 8 developers mention other reasons, such as studying projects and following repository advance.

Submitting pull requests covers 46 % of replies in the first-round survey, and the percentage rises to 79 % in the second round. Other reasons also cover higher percentage of developers in the second round. We will discuss impacts of predefined answers on validity in the Section 7.

As described in Table 1, submitting pull requests is one of main reasons of repository forking. However, submitting pull requests is only the superficial reason, and developers' goals are likely to be something else. We add a question to understand purposes of pull requests in the second-round survey. In the survey, we find that 128 developers fork repositories to submit pull requests, and 102 of them describe detailed purposes of them performing pull requests. We use open coding to come up with an inclusive set of purposes of pull requests. The first author reads 40 randomly selected replies, summarizes purposes of pull requests and builds codes of purposes. To validate the identified codes, the first, third and sixth authors apply codes on another 40 replies, compare their results, identify inconsistencies and agree on the final set of codes and purposes. At the end of the process, we divide the various purposes into 3 categories, and describe categories and their code sets in Table 2.

Table 2 shows that fixing bugs is the main purpose of pull requests, which covers 85.3 % of developers. Adding features is also mentioned by 45.1 % of developers. These results further confirm conclusions in Table 1: Developers fork repositories to fix bugs or add features. 10 developers submit pull requests to add documentation. For these developers, they fork repositories to submit pull requests and add documentation.

Table 2 Purposes of pull requests

Cause	Developers in the second round	Code set
Fix bugs	87 / 85.3 %	Fix bug, bugfix
Add features	46 / 45.1 %	Add feature
Documentation	10 / 9.8 %	Documentation

Table 3 Mechanisms through which developers find and fork repositories

Mechanism	Developers in the first round	Developers in the second round	Code set
External links	72 / 58.1 %	107 / 66 %	External link, web, blog, RubyGems, Hacker News, Reddit, Twitter, Facebook
Search	69 / 55.6 %	106 / 65.4 %	Google, search result
Friend	19 / 15.3 %	69 / 42.6 %	Follow, friend, word mouth
Recommendation	3 / 2.4 %	30 / 18.5 %	Recommendation
Other	N/A	25 / 15.4 %	N/A

3.3 Discovery Mechanisms

In the questionnaire that we send to developers to understand why they fork repositories, we also ask developers on how they discovered repositories that they forked. In the first round, we receive 124 replies and analyze these replies qualitatively. We also use open coding to come up with a set of mechanisms through which developers find repositories. We follow the same process that we describe in Section 3.2. At the end of the process, we divide main discovery mechanisms into 4 categories; these categories and their code sets are described in Table 3. The second column in Table 3 shows the number of developers that mention different discovery mechanisms in the first round. Since a developer may mention several discovery mechanisms, the sum of percentages in Table 3 is bigger than 100 %. From the results, we can note that:

1) 72 respondents discover repositories from external sites. More specifically, 22 respondents find repositories from Twitter, which is a popular social network and disseminates different kinds of information including software related contents (Tian et al. 2012). 20 respondents mention Hacker News or Reddit, which are social news websites and deliver content related to computer science and entrepreneurship; 5 respondents discover new repositories via RubyGems.[6] Ruby community is active on GitHub, and many GitHub repositories are primarily written using Ruby. Ruby's developers often browse RubyGems to discover interesting repositories that they can fork from GitHub.
2) The search mechanism covers 55.6 % of the respondents. Search engines provided by Google and GitHub are the most popular means for developers to search a repository of interest. 31 developers describe that they often use Google to find repositories; GitHub search engine is mentioned by 18 respondents. These people often have clear requirements on what repositories they want and search for specific repositories satisfying these requirements.
3) GitHub is a social coding site, and integrates social media functionality with code management tools (Jiang et al. 2013). Developers *follow* some interesting users, make friends and find new repositories via word of mouth. In GitHub, information about repositories that are created or forked by a developer will be broadcasted to developers that *follow* him/her.

[6] https://rubygems.org/

Table 4 Do developers value a recommendation tool?

Response	Developers in the first round	Developers in the second round
Yes	65 / 52.4 %	68 / 42 %
No	55 / 44.4 %	94 / 58 %
Unsure	4 / 3.2 %	N/A

4) Recommendation systems are popular and they have been applied to a variety of areas, such as movies, music, news or books. GitHub officially provides the explore page,[7] which implements a simple recommendation system. It recommends trending repositories, featured repositories and other popular repositories. Current system does not consider developer's interests or make personalized recommendation. Only 3 respondents find and fork repositories using this default GitHub recommendation system. Currently, GitHub's recommendation is rarely followed by developers. In the next subsection, we study developers' view towards repository recommendation.

In the second round, the question had predefined answers, which were based on results in the first round. The third column in Table 3 shows the number of developers who choose discovery mechanisms in the second round. Results in the second round confirm results in the first round. External links and search results are two main mechanisms through which developers find and fork repositories. Only 18.5 % of developers use recommendation to find repositories. 25 developers mention other discovery mechanisms: The word 'library' is mentioned by 5 developers, who use libraries and fork their repositories. Some other external links are mentioned, such as DailyJS and Gitter. In Table 3, discovery mechanisms cover higher percentage of developers in the second round than those in the first round. We will discuss impacts of predefined answers on validity in the Section 7.

3.4 Attitude Towards Recommendation

In our second-round survey, we ask developers whether they value a recommendation tool to help them find interesting repositories to fork. Table 4 shows results for the first and second rounds, respectively. In the first round, 52.4 % of respondents are positive, while 44.4 % of respondents are negative; in the second round, 42 % of respondents are positive, while 58 % of respondents are negative. In both rounds, more than 42 % of developers value a recommendation tool, while more than 44.4 % of developers do not value a recommendation tool.

In the second round, we ask developers to provide detailed reasons of their choices. More specifically, we ask developers two questions: If the recommendation tool is useful, what features of recommendation tool would developers like to have? If the recommendation tool is useless, why? For each question, the first, the third and the sixth authors independently read 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

[7] https://github.com/explore

conflicts and make classification. We do not use code sets here, because developers use different words to explain the same reasons.

In the second-round survey, a recommendation tool is appreciated by 68 respondents. Table 5 shows desired features of such recommendation tool. From the results, we can note that:

1) Ten developers hope that the tool recommends repositories which are similar to their previous repositories. For example, a respondent says that "It would be nice if it took into consideration the repositories I already forked." A desirable recommendation is one which is personalized, and considers historical records of repositories forked before.
2) Eight developers point out a specific feature of repositories that the recommendation tools should take note of, namely the programming language. The tool is expected to find repositories which are written in the same programming languages as those used in the developers' previous repositories. For example, a respondent says that "It should recommend code that it's written in the same programming language that I usually push to github."
3) Software quality is mentioned by 8 developers. For example, a developer mentions that "Code quality, which license, unit tests, and test coverage", and another developer mentions that "Better bug solving". These developers like to have a tool that recommends repositories of software of high quality (e.g., high test coverage and low bug count).
4) Five developers hope that the tool recommends popular repositories. For example, a respondent mentions that "Recommend initial repo and most visited forks".
5) Repositories with new technology are desirable for four developers. For example, a reply is "Recommend repositories based on the latest technologies one recently worked." The recommendation of new technology should also be personalized, and consider developers' interests.
6) Sixteen developers mention other features, and seventeen developers do not fill in detailed features of the recommendation tool.

In the above analysis, different developers have various requirements of the recommendation tool. But many developers emphasize the importance of personalized recommendation. Unfortunately, the current GitHub default recommendation system is not personalized.

In the second-round survey, a recommendation tool is deemed useless by 94 respondents, and Table 6 shows their detailed reasons. From the results, we can note that:

1) The most common reason is because forking is not the goal. As described in the Section 3.2, developers fork repositories to submit pull requests, fix bugs, add new features and

Table 5 Desired features of the recommendation tool

Feature	Developers
Similarity with previous repositories	10 / 14.7 %
Programming language	8 / 11.8 %
Quality	8 / 11.8 %
Popularity	5 / 7.4 %
New technology	4 / 5.9 %
Other	16 / 23.5 %
Not filled	17 / 25 %

Table 6 Why is the recommendation tool useless?

Reason	Developers
Forking is not the goal	34 / 36.2 %
Developers already know repositories before forking them	22 / 23.4 %
Developers do not need to have more repositories	8 / 8.5 %
It is difficult to recommend repositories	5 / 5.3 %
The number of forks are not good indicator	5 / 5.3 %
Other	13 / 13.8 %
Not filled	7 / 7.4 %

keep copies etc. Forking is the means instead of the final objective. Therefore, these developers do not go around looking for repositories to fork.

2) Twenty two or 23.4 % of the developers have already known which repositories to fork before forking them, and thus a recommendation tool is deemed useless. For example, a developer says that "I don't look for repos to fork, I fork tools I already need/use". These developers have definite requirements for repositories, and they do not need the recommendation.
3) Eight developers do not want to have more repositories, and thus they have negative attitude toward a recommendation tool. One example of their responses is: "I fork repos I work with I dont need to go find new things to add more to my work load."
4) Five developers think that it is difficult to recommend repositories. For example, a developer writes that "I think that it will contain a lot of irrelevant information, which will make the tool useless after a while." If a recommendation tool is good enough, some developers may change their minds.
5) Another 5 developers think that a recommendation system based on the number of forks will not work. A developer responds: "I don't care about how many forks there are, that's sometimes even an indication that the project isn't well maintained anymore. I care about the functionality the code provides."
6) Thirteen developers mention other reasons, and seven developers do not fill in detailed reasons.

3.5 Attitude Towards Repository Owner

In the second-round survey, we ask developers whether they care about who the repository owners are, when developers fork repositories. Table 7 shows the results. 35.2 % of respondents care about repository owners, while 60.5 % of respondents do not care about repository owners. 4.3 % of respondents are unsure.

We ask developers to provide detailed reasons of their choices. We follow the same process described in the Section 3.4, and classify reasons into corresponding categories. Fifty

Table 7 Do developers care about who repository owners are?

Response	Developers
Yes	57 / 35.2 %
No	98 / 60.5 %
Unsure	7 / 4.3 %

Table 8 Reasons why developers care about repositories owners

Reason	Developers
Active in reviewing and accepting pull requests	15 / 26.3 %
Good reputation	13 / 22.8 %
Friend	7 / 12.3 %
Long-term maintainer	3 / 5.3 %
Other	19 / 33.3 %
Not filled	3 / 5.3 %

seven developers care about repository owners, and Table 8 shows detailed reasons. From the result, we can note that:

1) Fifteen developers care about repository owners, because they want to make sure that repository owners are active in reviewing and accepting pull requests. For example, a developer says that "because its important to know how quickly they will react and how "reasonable" they are with dealing with contributions" Developers do not like to waste their time if repository owners seldom look at pull requests from external developers.
2) Good reputation is mentioned by 13 developers. They are more likely to fork repositories if owners are well known. In GitHub, the number of followers is an important indicator of reputation (Lee et al. 2013). In Section 5.2, we will study the number of followers of repository owners.
3) Seven developers care about repository owners because they mainly fork repositories from friends. As shown in Table 3, recommendation through friends is a mechanism through which developers find and fork repositories.
4) Three developers fork repositories from owners who are long-term maintainers of repositories. For example, a developer says that "mainly because I care about if the project will be maintained, and I want to know who people are if they are prominent contributors to the community."
5) Nineteen developers mention other reasons, and 3 developers do not fill in detailed reasons.

Ninety eight developers do not care about repository owners, and Table 9 shows detailed reasons. From the result, we can note that: Firstly, 19 developers care about the code in the repositories, rather than their owners. Secondly, 4 developers do not care about specific owners as long as the repositories are actively maintained. Thirdly, 4 developers mainly consider licenses when they fork repositories. Finally, 18 developers mention other reasons, and 53 developers do not fill in detailed reasons.

Table 9 Reasons why developers do no care about repository owners

Reason	Developers
Code	19 / 19.4 %
Activeness	4 / 4.1 %
License	4 / 4.1 %
Other	18 / 18.4 %
Not filled	53 / 54.1 %

In this subsection, we explore whether developers care about owners when they fork repositories. Results show that repository owners are taken in consideration by 35.2 % of respondents. We also analyze their detailed reasons, which inspire us to make deeper analysis. In Section 5.2, we make an initial effort and study some attributes of repository owners, such as the number of followers. In future, we will study more features of repository owners.

> Developers fork repositories to submit pull requests, fix bugs, add new features and keep copies etc. Developers find repositories to fork from various sources: search engines, external sites and friends, etc. Though less than 18.5% of respondents find repositories via recommendation, at least 42% of respondents describe that recommendation tool will be useful to know and fork repositories. 35.2% of respondents care about repository owners when they fork repositories.

4 What Developers Fork?

In this section, we mainly study what kinds of repositories developers fork. More specifically, we explore which programming languages developers use in forked repositories. The choice of programming languages reflects preference towards some kinds of tasks (FBissyande et al. 2013). For example, Yacc is mainly used in the development of compilers, while PHP is applied in the development of web applications. The thorough analysis can guide GitHub to recommend new repositories and meet developers' preference.

We characterize each repository by the primary programming language that is used to write code stored in it, and study which programming languages GitHub developers prefer. Though a repository may have code written in various languages, GitHub's API returns the major (i.e., primary) programming language for each repository. The major programming language is the most representative language and is the one used in our analysis.

Our initial step is to explore how many programming languages each developer uses. We compute the number of programming languages in forked repositories and plot the results in Fig. 4: 27.3 % of the developers only use 1 programming language in forked repositories, and 28.2 % of the developers fork repositories written in 2 programming languages. For 92 % of developers, the number of languages in forked repositories is at most 5. The majority of developers have obvious preference towards a small number of programming languages. This is probably because most of the developers are familiar with a few

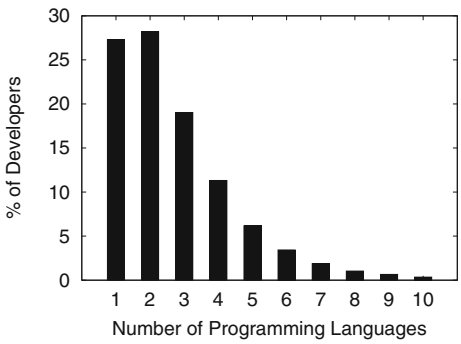

Fig. 4 Number of programming languages

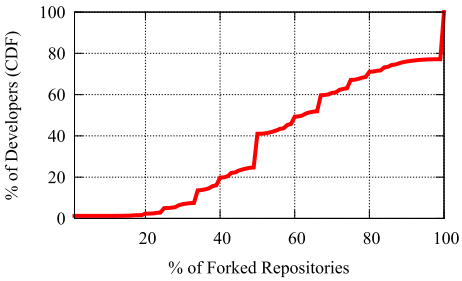

Fig. 5 Percentage of forked repositories written in primary programming language

languages, which they often use in software development. It is convenient for developers to read and modify codes written in their familiar languages, and thus they seldom fork repositories with codes written in unfamiliar programming languages. Moreover, the choice of programming languages reflects the preference towards some kinds of tasks. Developers have special interests in some programming tasks, and always use corresponding languages.

Developers only use a few programming languages in forked repositories. Are these programming languages used with similar or different frequency? Does one programming language play a key role in the software development? For every developer, we define his/her *primary programming language* as the language most frequently used in his/her forked repositories; Then we compute the ratio of the number of forked repositories written in this primary programming language to the total number of forked repositories of this developer. We aggregate these ratios across all the developers and plot the results in Fig. 5. For only 19.8 % of developers, the primary programming language covers less than 40 % of the forked repositories; For the other 80.2 % of developers, the primary programming language covers more than 40 % of the forked repositories; For 50.7 % of developers, the percentage of forked repositories written in the primary programming language is even larger than 60 %. Surprisingly, our results indicate that most of developers mainly fork repositories written in only 1 programming language, and they do not like to use other programming languages. GitHub has a large number of repositories written in different programming languages (FBissyande et al. 2013), and provides a good opportunity for developers to use other programming languages. However, developers do not make good use of this opportunity, and still stick with the most familiar language. To improve GitHub's current repository recommendation system, GitHub should make its recommendations more personalized by considering the preference of individual developers (e.g., their primary programming languages).

Developers often use one primary programming language in forked repositories. We would like to know which language it is. For each programming language, we compute the number of developers who consider it as a primary, divided by the total number of developers. Figure 6 shows the distribution of developers for the top ranked languages. Overall, JavaScript appears to be the most popular language among developers, and it is the primary programming language for 21.9 % of the developers. The second most popular language is Ruby, followed by Java, Python and PHP. The ten programming languages shown in Fig. 6 together cover 89.1 % of developers in GitHub.

Developers are likely to fork repositories written in their preferred programming languages.

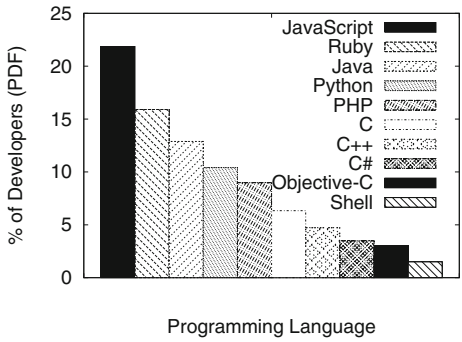

Fig. 6 Percentages of developers who primarily use a programming language

5 From Whom Developers Fork Repositories?

Developers fork repositories from various parent repositories owned by other developers. Some of these owners are repository creators while other owners also forked their repository from others in the past. Some repository owners are popular, and attract many developers to fork their repositories and make contributions; other repository owners are unpopular and their repositories rarely attract forks. In this section, we study and characterize repository owners and explore how they impact developers likelihood to fork their repositories. More specifically, we mainly study the following questions: What is the composition of creators among owners of repositories that developers fork? What are the differences between popular and unpopular repository owners? Answers to these questions can guide GitHub to find repositories to recommend based on the characteristics of their owners.

5.1 Composition of Repository Owners

In this subsection, we want to figure out the composition of repository owners that are forked by other developers. We pose two questions: First, what portion of owners are repository creators? Second, do developers fork repositories from the same repository owner, or different repository owners?

We begin by addressing the first question. To find an answer to this question, for each forked repository, we check whether the owner of its parent repository is the creator or not. If the owner is the creator, its parent repository and source repository (i.e., root repository in a fork tree – see Section 2.2) are the same. Next, for each developer, we compute the number of his/her forked repositories that are directly forked from the creators, divided by the total number of forked repositories that the developer has. Figure 7 plots the distribution of the percentage of repositories directly forked from creators. Only 3.9 % of developers fork less than 60 % of their forked repositories directly from creators, and the other 96.1 % of developers fork more than 60 % of their forked repositories directly from the creators. The end of the curve has a sudden jump since 68.4 % of developers fork all their repositories directly from creators. In our datasets, 68.6 % of forks come from 10 % of repositories. This follows the Pareto principle, which is often observed in many datasets. Some repositories may be owned by non-creators, but they are primary branches. We mainly explore whether developers fork repositories directly from creators. Developers may fork primary repositories from non-creators, which is beyond the scope of this paper.

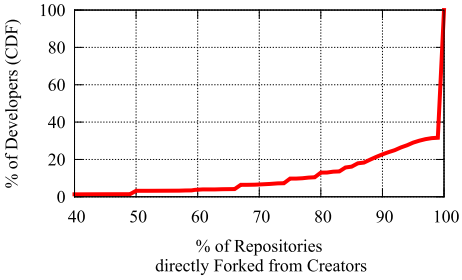

Fig. 7 Percentage of repositories directly forked from creators

In GitHub, a repository can be several "hops" away from its source repository in its fork tree. Consider four repositories A, B, C, D, where repository B is forked from repository A, repository C is forked from repository B, and repository D is forked from repository C. The owner of repository C may modify code greatly before it gets forked to form repository D. Code contained in repository C and repository A can be very different. Therefore, forking may produce incompatible and competing versions, and this has been considered to be a danger to OSS development (Fung et al. 2012). However, our results show that most of forked repositories are just one hop away from its source repository. The majority of developers fork from a repository that is owned by a creator. Repository forking provides developers good opportunity to freely modify code and make individual decisions, but generally does not split developers to work on different competing and incompatible versions of repositories.

In GitHub, developers fork repositories and makes changes. Then developers submit pull requests when they want to merge changes into main repositories. Forked repositories may also be updated and synchronized with original repositories. Figure 7 shows that the majority of developers fork from a repository that is owned by a creator. We take a further step, and explore whether developers update forked repositories and submit pull requests back into the original repositories.

Popular repositories attract many forks, and they are worthy of this further exploration. We choose 3 popular repositories (i.e., *zendframework/zf2*, *scala/scala* and *xbmc/xbmc*). The project *zendframework/zf2* is developing a framework for building modern, high-performing PHP applications; the project *scala/scala* is built for developing the Scala programming language; the project *xbmc/xbmc* is developing an open source software media player and entertainment hub for digital media. In GitHub, starring allows users to keep track of projects that they find interesting. These 3 repositories all have more than 4,000 stars, and thus they are highly popular in GitHub.

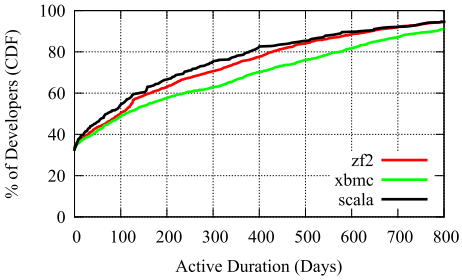

Fig. 8 Time interval between forked time and updated time

Table 10 Percentage of developers who update the forked repository and submit pull requests

Repository	% of developers
zendframework/zf2	31.6 %
xbmc/xbmc	31.3 %
scala/scala	48.4 %

We collect pull requests of these 3 repositories in GitHub, and analyze repositories forked from them. For each of the forked repositories, we collect the time it was forked and its last update time. We define an active duration of a forked repository as the time interval between the time a repository is forked and its last update time. For each of the 3 popular repositories, and each developer who fork it, we aggregate the active duration across all their forked repositories and plot the results in Fig. 8. From the results, we can notice that for more than 60 % of the developers, their forked repositories eventually get updated (active duration ≥ 0).

For these developers who have updated their forked repositories, we want to investigate the percentage of them who have issued pull request. We analyze the pull requests for the 3 repositories and extract the submitters of these pull requests. Based on these submitters, we construct a set of developers who ever submit pull requests. Next, we compute the number of developers who ever update their forked repository and submit pull requests, divided by the number of developers who ever update their forked repositories. Table 10 shows the ratios for the 3 popular repositories. In repositories *zendframework/zf2*, *xbmc/xbmc* and *scala/scala*, 31.6 %, 31.3 % and 48.4 % of developers submit pull requests after they update repositories. Other developers update repositories but never submit pull requests. One of reasons is that some developers only synchronize updates between the forked repository and the original repository, but they do not make any modification.

Next, we examine the likelihood that developers will repeatedly fork several repositories from the same owner. As described in the Section 2.2, we crawled datasets in October 2013. In order to explore how repository forking changes as time goes by, we randomly selected 11,015 developers, and crawled their information again in May 2015. In total, we collected 11,015 developers and their 160,463 forked repositories. For each developer, we compute the number of his/her forked repositories, divided by the number of unique owners of parent repositories. Figure 9 plots the average number of repositories forked per owner, which are based on datasets crawled in October 2013 and May 2015, respectively. We note that: In October 2013 line, 67.5 % of developers fork repositories from an owner only once, and 93.4 % of developers fork on average less than 1.4 repositories from an owner. In May 2015 line, 61.2 % of developers fork repositories from an owner only once, and 95.3 % of

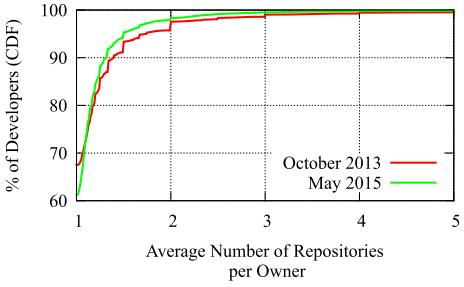

Fig. 9 Average number of repositories per owner

developers fork on average less than 1.4 repositories from an owner. Results of our separate analysis using data from October 2013 and May 2015 are similar. Surprisingly, results indicate that developers seldom fork multiple repositories from an owner.

5.2 Attributes of Repository Owners

Figure 9 shows developers mostly fork repositories from different owners. Therefore, developers are unlikely to fork additional repositories from owners of repositories that they have forked before. Though owners of previous forked repositories are not very useful in predicting owners of repositories that will be forked in the future, they are useful to discover the characteristics of owners who attract forks. In this subsection, we analyze attributes of repository owners who are attractive and their repositories get forked. Results can potentially be used by GitHub to recommend repositories created by a particular group of owners.

To understand the level of attraction of different owners, we sort all owners by the number of forks their repositories attract. Then we plot the distribution of forks among owners ordered by the number of forks in Fig. 10. The bulk of forks can be attributed to a very small portion of owners: For the top 5 % of owners, their repositories attract 65.1 % of forks; For the other 95 % of owners, their repositories attract 34.9 % of forks. It shows that the distribution of forks is not equally distributed among the owners. Developers prefer to fork repositories from a minority of attractive owners. Based on these findings, we classify owners into two groups: the top 5 % of owners are defined as *attractive owners*, and other owners are defined as *unattractive owners*. In the following paragraphs, we compare characteristics of attractive and unattractive owners. Note that this classification is mainly used to understand characteristics of attractive repository owners. We also observe similar results, when we try some other thresholds in the definition of attractive/unattractive owners, e.g., the top 10 % of owners as attractive owners.

Our first analysis is to study the GitHub account type of repository owners. GitHub offers two account types, namely personal and organization. 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. We compute the percentages of organization accounts and personal accounts that are attractive owners. We also compute similar percentages for unattractive owners. We plot the results in Fig. 11. We find that 19.5 % of the attractive owners are organizations, while only 4.9 % of the unattractive owners are organizations. Attractive owners contain a higher percentage of organization accounts than unattractive owners. This is the case since organizations often develop better quality open

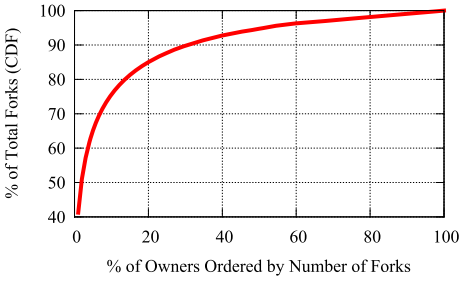

Fig. 10 Distribution of forks, with owners ordered from most to least attractive

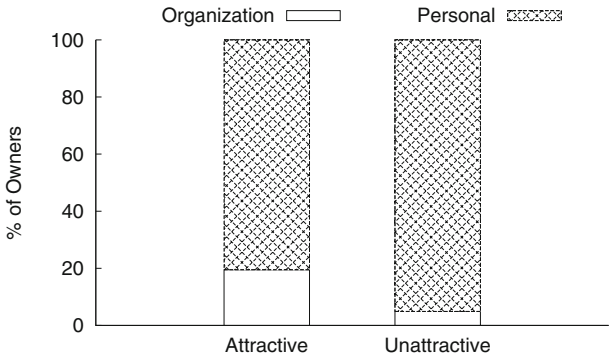

Fig. 11 Type of attractive and unattractive owners

source projects than individual developers. Our findings indicate that, for GitHub's recommendation, repositories owned by organizations should be assigned higher priorities than those owned by personal accounts.

GitHub integrates social media functionality with code management tools (Jiang et al. 2013). In GitHub, one developer can *follow* another developer and receive updates of activities of the developer that he/she follows. In our second analysis, we explore how follow links in GitHub's social network are linked to the attractiveness of repository owners. To do this, we count the in-degree and out-degree of each repository owners. The in-degree is the number of followers that the repository owner has, and the out-degree is the number of developers that are followed by the repository owner.

Figure 12 shows the distribution of out-degrees for attractive and unattractive owners. The boxplots are drawn using the SPSS analysis tool. Some points are identified as outliers by the SPSS tool, and they are excluded from the analysis. We manually check these outliers, and find that they are abnormal owners. For example, the developer *equus12* is

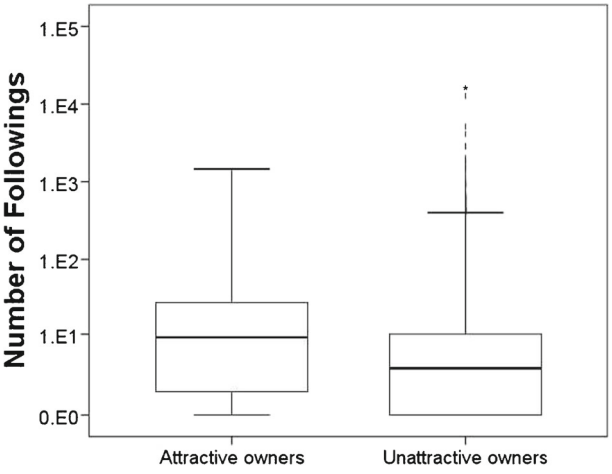

Fig. 12 Out-degree distribution of attractive and unattractive owners

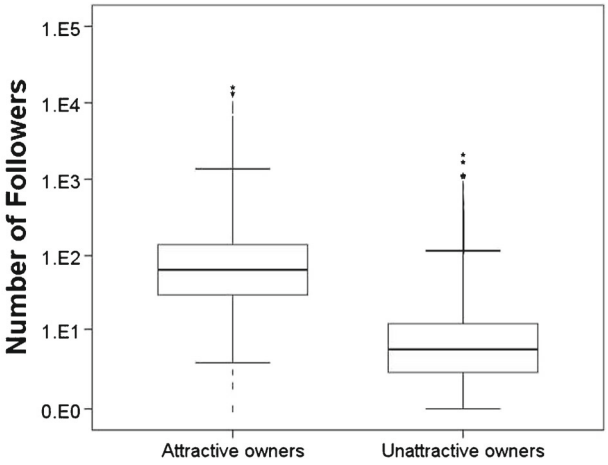

Fig. 13 In-degree distribution of attractive and unattractive owners

an unattractive owner.[8] This developer has 2 forked repositories, but follows 12,686 developers. This strange developer is inactive in the software development, but he is extremely active in following developers. Therefore, the developer *equus12* is identified as an outlier, and he is excluded from the analysis.

From the boxplots in Fig. 12, we find that attractive owners have larger out-degrees than unattractive owners. Attractive owners have a median out-degree of 9, while unattractive owners have a median out-degree of 3. The lower quartile is 0 for unattractive owners, which is the same as the minimum value. The Mann-Whitney-Wilcoxon (MWW) test is a non-parametric statistical test that assesses the statistical significance of the difference between two distributions (Mann and Whitney 1947). Using the MWW test, we have tested and confirmed that the difference between attractive and unattractive owners is statistically significant at 0.001 significance level.

Figure 13 plots the distribution of in-degrees for attractive and unattractive owners. The SPSS tool automatically identifies some points as outliers, and excludes them from the analysis. We manually check these outliers, and find that they are mainly caused by 2 reasons. Firstly, the in-degree is abnormal for some owners. We still take the developer *equus12* as an example. This developer has as many as 606 followers, but has only 2 forked repositories. It remains unknown why so many developers follows him. The in-degree is strange and thus this developer is identified as an outlier. Secondly, the in-degree is extremely high for some special developers. For example, the developer *mojombo* has 17,902 followers, because he is the founder and former CEO of GitHub. His in-degree is much higher than others, and he is identified as an outlier. These outliers are automatically identified and excluded from the analysis.

In Fig. 13, we also find that attractive owners have much larger in-degrees than unattractive owners. The median value of in-degree is 65 for attractive owners, while the median value of in-degree is only 5 for unattractive owners. We also use the MWW test to compare the in-degrees of attractive and unattractive owners, and we get a p-value of $2.3e^{-271}$. This

[8] http://github.com/equus12

means that attractive owners have statistically significantly more followers than unattractive owners. In GitHub, developers freely choose interesting developers and follow them. High in-degree developers are highly likely to be developers with good reputation (Lee et al. 2013). Therefore, GitHub should recommend repositories from developers with high in-degrees.

Finally, we study the age of attractive owners. The age of an account is defined as the number of months between its registration and the month when we collected our datasets. The larger the age is, the earlier the account registers in GitHub. In Fig. 14, we plot the age distribution of attractive and unattractive owners. The median age is 55 for attractive owners, while this number is only 43 for unattractive owners. We have done the MWW test and have found that the difference is statistically significant at the significance level of 0.001. Attractive owners are registered earlier than unattractive owners.

The majority of developers directly fork repositories from creators. Developers seldom fork multiple repositories owned by the same owner. In comparison with unattractive owners, attractive owners have higher percentage of organizations, more followers and earlier registration in GitHub.

6 Discussion

We compare our findings with previous works and discuss implications in this section.

6.1 Forking and Growth of Software Projects

The free software community has different attitudes towards forking. On the negative side, forking is considered as a danger to OSS development (Nagy et al. 2010; Muffatto and Faldani 2003). On the positive side, forking satisfies different kinds of needs, such as adding new features, fixing bugs and meeting commercial strategy (Robles and Gonzalez-Barahona 2012).

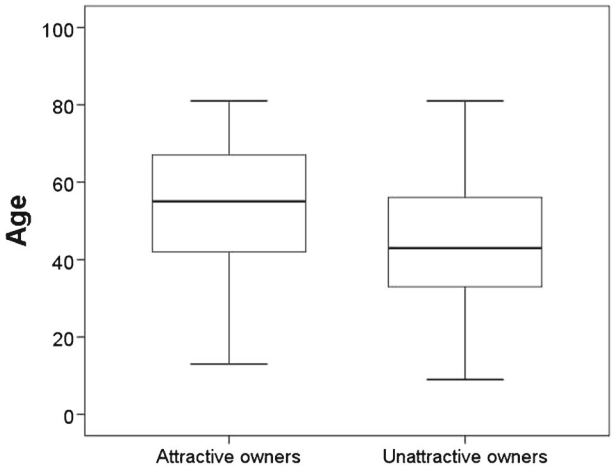

Fig. 14 Age of attractive and unattractive owners

Though forking is controversial in traditional OSS community, GitHub encourages frequent forking as a built-in feature of its infrastructure. Our results provide important insights for the OSS community and GitHub. Our qualitative results show that forking in GitHub does not split developers into different competing and incompatible versions of repositories. Instead, the main cause of forking is submitting pull requests and making contributions to the original repositories. Our quantitative results show that most of the developers directly fork repositories from original creators, and contribute to these original repositories (rather than repositories forked by others). Therefore, forking is beneficial for OSS development, and should be encouraged in OSS community (at least in GitHub, which is the largest hosting site of software projects). Repository owners should not be afraid to share their code with others as it would most likely lead not to competition but growth.

We have also performed a deeper analysis of forks in several highly-popular repositories and observe that for the majority of the developers, their forked repositories eventually get updated (see Section 5.1). Additionally, more than 30 % of developers submit pull requests after they update repositories. These results further demonstrate that repository forking is an indicator or even catalyst for future contributions. We thus suggest repository owners to actively attract developers to fork their repositories, and then encourage them to make future contributions.

6.2 Discovery Mechanism

Recruiting and retaining new contributors is a critical success factor for OSS projects (Crowston et al. 2012). Since the main cause of forking is making contributions to the original repositories, attracting forks is important for recruiting new contributors. In order to provide suggestions about best ways to attract forks, we send questionnaires to investigate how developers find repositories which they fork. We find that developers mainly find and fork repositories through external links, search engines, friends and the GitHub default recommendation. These results highlight ways repository owners can advertise their repositories to attract forks.

For example, we observe that the most common ways of developers to find repositories to fork is via external links (e.g., blog sites, etc.). Thus, it would be useful for developers to post their repositories in popular external sites. For example, we find that Ruby's developers often browse RubyGems[9] to discover interesting repositories that they fork from GitHub. When repositories are written in Ruby, their owners should use RubyGems to promote repositories. Moreover, up to 42 % of respondents find repositories to fork from friends. Therefore, we suggest that repository owners may introduce their repositories to some popular developers who have many followers, and fully utilize social relationships to advertise repositories and attract forks.

6.3 Repository Recommendation

Many researchers have built recommendation systems to help developers perform a number of software engineering tasks (Robillard et al. 2010; Happel and Maalej 2008). The field of recommendation systems for software engineering (RSSE) has grown substantially with a dedicated workshop (i.e., International Workshop on Recommendation Systems for Software Engineering) which has run 4 times since 2008 and a book written on the topic

[9] https://rubygems.org/

(Robillard et al. 2014). Unfortunately, to the best of our knowledge, there is little research study that builds systems for recommending repositories in GitHub. Currently, GitHub recommends trending repositories, or repositories starred by friends. A workshop paper by Zhang et al. presents a simple method of recommending relevant projects from GitHub user behavior data; unfortunately the average relevance score that their method achieves is still very low (Zhang et al. 2014). Thus more work is needed to build effective recommendation systems for repositories hosted in GitHub.

In this paper, to help advance work on recommendation systems for software engineering (in particular studies that recommend repositories in GitHub), we send questionnaires to understand developers' attitude towards such a recommendation system. This would validate or refute the idea of building more advanced recommendation systems. We find that more than 42 % of developers that we have surveyed agree that an automated recommendation tool is useful to help them pick repositories to fork, while more than 44.4 % of developers do not value a recommendation tool. These results suggest that building such a recommendation system would be valuable since almost half of the developers value it - admittedly, it is not for everyone though. This provides an empirical basis for supporting future work that builds advanced recommendation systems for GitHub. We strongly believe it is a good practice to talk to our potential "clients" (i.e., developers) before building tools that we claim useful for them.

Moreover, we take a step to understand desired features of the recommendation tool. We find that many respondents care about the similarity of a repository with previous repositories that they have forked, programming languages of the repository, quality of the code in the repository, popularity of the repository, new technology used by code in the repository, and the repository owner. These results reflect developer requirements, and provide insights for future researchers and GitHub to realize effective recommendation systems that can help developers and improve how they collaborate in GitHub.

We perform a deeper analysis of programming languages used and repository owners. We observe that developers are likely to fork repositories written in their preferred programming languages. Therefore, programming languages are useful as a preliminary filter of repositories to be recommended. Our results suggest that a recommendation tool does not need to consider owners of repositories which are forked by the developer before. Instead, the recommendation tool should consider the type and in-degree of repository owners. The tool should particularly consider repositories created by older organizations with more followers. GitHub or researchers working on recommendation systems for software engineering (RSSE) can consider these heuristics to build an effective system to recommend repositories in GitHub.

7 Threats to Validity

Threats to internal validity relates to experimenter biases and errors. Firstly, we use a set of scripts to download and process the large GitHub data. We have checked these scripts and fix errors that we found; however, there could still be errors that we do not notice. Secondly, we study developers who have at least 5 forked repositories. These developers only cover 31.6 % of all developers, but they have 84.8 % of all forked repositories. Our conclusions are applicable to developers who are active in forking repositories. Though the characteristics of inactive developers are not investigated in this work, they only have 15.2 % of forked repositories. Thirdly, some points are determined as outliers and excluded from the analysis of attractive owners and unattractive owners. This may lead to bias in the analysis.

Nonetheless, we manually check these outliers and find that they are mainly abnormal or special developers. Moreover, the number of outliers is very small in our dataset. Finally, we sent questionnaires and run 2 rounds of survey. We received replies from 124 developers in the first round, and received replies from 162 developers in the second round. These respondents were selected from 23,782 developers who had at least 30 forks. 124 respondents from a population of 23,782 developers yield a 90 % confidence level with a 7.37 % error margin; 162 respondents from a population of 23,782 developers yield a 90 % confidence level with a 6.44 % error margin.

Threats to external validity relates to the generalizability of our study. Firstly, we analyze 236,344 developers and 1,841,324 forks. We believe that these are large numbers that can mitigate the threat to external validity. Still, we only analyze developers, forks, and repositories in GitHub. It is not clear if our results will generalize to other open source platforms. In the future, we plan to study a similar set of research questions using developers, forks, and repositories from other platforms such as Bitbucket, and compare the results with the results that we find for GitHub. Secondly, our study is limited to GitHub and it may not represent what developers do in conventional OSS development where forking *usually* entails a more serious underlying community process (usually, some form of disagreement) and the two forks diverge significantly. In GitHub, there are cases of such divergences but they are not the majority. Thirdly, we only analyze 3 popular repositories, and explore whether developers update forked repositories and submit pull requests back. Analysis results might not be generalizable to all repositories as we only analyze 3 repositories in this study.

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 the questionnaire. In the first-round survey, we read 124 developer replies and manually build code sets of forking reasons and discovery mechanisms. We must admit that this process is a subjective one. In order to reduce human errors, two authors independently build codes for forking reasons and discovery mechanisms. Then two authors compare their results and agree on the final set of codes. In the second-round survey, we read 162 developer replies and manually classify reasons about their attitudes toward recommendation tool and repository owners. This process is also a subjective one. To try to reduce human errors, three authors instead of two analyze the replies and independently make classification. Secondly, we ask developers about whether they would like a recommendation tool in GitHub. However, replies only shows developers' initial impression of such recommendation tool. A much better way to obtain developers' input is to implement such a recommendation tool, ask developers to try it, and then gather their opinions. The construction of such recommendation tool though is beyond the scope of this paper.

Construct validity threats are related to the degree to which the construct being studied is affected by experiment settings. Firstly, a frequently observed threat on a questionnaire-based analysis is that designed questions may misguide responders. We have tried to reduce bias, e.g., by introducing the option "other" and not restricting responses to only a few answers. However, researcher bias may still influence the wording of questions, and lead respondents to particular answers. Secondly, in the second-round survey of forking reasons and discovery mechanisms, we design predefined answers, and provide multiple choices. These predefined choices may impact developers' replies. However, these predefined choices are based on results in the first round. We include an optional "other" response, and allow respondents to write other answers. Thirdly, we mainly study programming languages and owners of repositories. In the future, we will study more properties of

repositories, such as such as implemented functionalities, software quality (e.g., test coverage and bug count), and developers' activeness in repositories. We will explore how repository properties influence developers' choices when they fork repositories. Finally, we have not studied all potential metrics for characterizing and differentiating attractive and unattractive owners. In future work, we will study some other metrics, such as project quality and developer experience.

8 Related Work

Forking has been widely studied among researchers that analyze open source software (OSS). The OSS community has different attitudes towards forking. On the negative side, Muffatto and Faldani believe that forking dilutes the community of open source software as the average number of developers per project decreases (Muffatto and Faldani 2003). Nagy et al. mention that forking can be considered as a danger to OSS development, because forking introduces duplication of effort, reduces communication and may produce incompatible and competing versions (Nagy et al. 2010). DiBona et al. find that individuals and companies are afraid of losing control if forking is allowed (DiBona et al. 1999). Neville-Neil argues that only abandoned projects should be forked as a developer who forks may be taken as a petulant and spoiled child (Neville-Neil 2011). On the positive side, the beauty of OSS development is that no permission from the authors is needed to start a fork. Nyman et al. observe that code forking increases potential innovation by allowing for the combination and modification of OSS projects (Nyman and Lindman 2013). Ernst et al. and Robles et al. find that forking satisfies different kinds of needs, such as adding new features, fixing bugs, etc. (Ernst et al. 2010; Robles and Gonzalez-Barahona 2012). Ernst et al. analyze the forks of one software project named Trac, while Robles et al. analyze several hundreds of forks. Similar to Ernst et al. and Robles et al., we also analyze the reason why developers fork repositories. We extend their study by sending questionnaires to developers that forked various projects in GitHub and analyzing 124 replies and 162 replies in two rounds. Different from our study, these studies do not analyze repositories in GitHub. Also, our study answers a number of research questions that are not investigated by these existing studies.

Fung et al. performs a preliminary analysis of social forking in GitHub (Fung et al. 2012). They investigate 9 OSS communities that develop projects written in JavaScript. They investigate different kinds of forks that are made (e.g., endogenous vs. exogenous, primary vs. secondary), the number of times developers perform forking, and the different kinds of contributions that these developers make to the project that they fork from. In this work, we extend their preliminary study in various ways: First, we investigate many additional research questions. Our research questions analyze why and how developers perform forking, what developers fork, and from whom developers fork. Second, we increase the scale of their study many times larger; we investigate 236,344 developers and their 1,841,324 forks of projects written in various programming languages. Fung et al. only analyze "close to seven thousand developers" who "created about eight thousand forks" of projects written in one programming language (i.e., JavaScript). Third, we not only analyze data downloaded from GitHub but also send questionnaires to real developers.

Recently, Gousios et al. perform an exploratory study of pull-based software development model in GitHub (Gousios et al. 2014). They investigate the popularity of pull-based development model, factors that affect the decision to merge a pull request, and factors that affect the time it takes to process a pull request. In GitHub, developers fork repositories,

make modification and then submit pull request. Forking is the initial step of submitting pull requests. Different from Gousios et al., we focus on forking rather than pull requests, and analyze a new set of research questions on why and how developers fork, what they fork, and from whom they fork. Thus, our studies complement each other.

Bissyande et al. investigate the programming languages that are used to develop projects in GitHub (FBissyande et al. 2013). In this work, we extend their study by highlighting that developers often fork projects written in their primary programming language. We also investigate many other research questions. Many other studies analyze GitHub data from different angles and perspectives: testing culture (Pham et al. 2013), transparency and collaboration (Dabbish et al. 2012, 2013), pull requests (Gousios et al. 2014), recruitment (Marlow and Dabbish 2013), project success (Tsay et al. 2012), social properties (Begel et al. 2013; Thung et al. 2013), etc. Different from these studies, we investigate a new perspective by answering a new set of research questions.

9 Conclusion and Future Work

This paper presents an empirical study of why and how developers fork what from whom in GitHub. We show empirical evidence that:

(1) (Why) Developers fork repositories to submit pull requests, fix bugs, add new features and keep copies etc. (2) (How) Developers find repositories to fork from various sources: search engines, external sites (e.g., Twitter, Reddit), social relationships, etc. (3) (What) Developers are likely to fork repositories written in their preferred programming languages. (4) (From Whom) Developers mostly fork repositories from creators. In comparison with unattractive owners, attractive owners have higher percentage of organizations, more followers and earlier registration in GitHub. Our datasets and survey results are publicly available, and they can be downloaded from the project homepage.[10]

In the future, we plan to extend our empirical study to other OSS platforms, e.g., BitBucket. We also plan to investigate typical profiles of developers by automatically characterizing the types of repositories that they fork. Using these profiles, we plan to develop a personalized recommendation tool. It is interesting to investigate when and why some forks cause disagreement within the software development communities, and split up of repositories into competitive and incompatible versions, which is also left as a future work.

Acknowledgments This work is supported by National Natural Science Foundation of China under Grant No.61300006, the State Key Laboratory of Software Development Environment under Grant No.SKLSDE-2015ZX-24, and Beijing Natural Science Foundation under Grant No.4163074.

References

Begel A, Bosch J, Storey MA (2013) Social networking meets software development: perspectives from github, msdn, stack exchange, and topcoder. IEEE Soft 30(1):52–66

[10] https://github.com/lightbot/ForkResearch

Bird C, Rigby PC, Barr ET, Hamilton DJ, German DM, Devanbu P (2009) The promises and perils of mining git. In: Proceedings of MSR, Vancouver
Crowston K, Wei K, Howison J, Wiggins A (2012) Free/libre open source software development: What we know and what we do not know. ACM Comput Surv:44
Dabbish L, Stuart C, Herbsleb J (2012) Social coding in github: transparency and collaboration in an open software repository. In: Proceedings of CSCW, Washington
Dabbish L, Stuart C, Tsay J, Herbsleb J (2013) Leveraging transparency. IEE Soft 30(1):37–43
DiBona C, Ockman S, Stone M (eds) (1999) Open sources: voices from the open source revolution. O'Reilly
Ernst NA, Easterbrook S, Mylopoulos J (2010) Code forking in open-source software: a requirements perspective. arXiv:1004.2889
FBissyande T, Thung F, Lo D, Jiang L, Reveillere L (2013) Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In: Proceedings of COMPSAC, Kyoto
Fung KH, Aurum A, Tang D (2012) Social forking in open source software: an empirical study. In: CAiSE forum, Poland
Gousios G, Pinzger M, van Deursen A (2014) An exploratory study of the pull-based software development model. In: ICSE, Hyderabad
Happel HJ, Maalej W (2008) Potentials and challenges of recommendation systems for software development. In: Proceedings of the international workshop on Recommendation systems for software engineering, pp 11–15
Jiang J, Zhang L, Li L (2013) Understanding project dissemination on a social coding site. In: Proceedings of WCRE, Koblenz
Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining github. In: Proceedings of MSR, Hyderabad
Lee MJ, Hahn J, Ferwerda B, Moon JY, Choi J, Kim J (2013) Github developers use rockstars to overcome overflow of news. In: Proceedings of CHI, pp 133–138
Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. Ann Math Stat 18(1):50–60
Marlow J, Dabbish L (2013) Activity traces and signals in software developer recruitment and hiring. In: San Antonio
Muffatto M, Faldani M (2003) Open source as a complex adaptive system. EMERGENCE 5(3):83–100
Nagy D, Yassin A, Bhattacherjee A (2010) Organizational adoption of open source software: barriers and remedies. Commun ACM 53(3):148–151
Neville-Neil GV (2011) Think before you fork. Commun ACM 54(6):34–35
Nyman L, Lindman J (2013) Code forking, governance, and sustainability in open source software. Technology Innovation Management Review:7–12
Pham R, Singer L, Liskin O, Filho FF, Schneider K (2013) Creating a shared understanding of testing culture on a social coding site. In: Proceedings of ICSE, San Francisco
Robillard MP, Walker RJ, Zimmermann T (2010) Recommendation systems for software engineering. IEEE Soft 27(4):80–86
Robillard MP, Maalej W, Walker RJ, Zimmermann T (2014) Recommendation systems in software engineering. Springer
Robles G, Gonzalez-Barahona JM (2012) A comprehensive study of software forks: Dates, reasons and outcomes. Open Source Systems: Long-Term Sustainability 378:1–14
Thung F, FBissyande T, Lo D, Jiang L (2013) Network structure of social coding in github. In: 17th European conference on software maintenance and reengineering, Genova
Tian Y, Achananuparp P, Lubis IN, Lo D, Lim EP (2012) What does software engineering community microblog about? In: MSR, pp 247–250
Tsay J, Herbsleb J, Dabbish L (2012) Social media and success in open source projects. In: Proceedings of CSCW, Seattle
Zhang L, Zou Y, Xie B, Zhu Z (2014) Recommending relevant projects via user behaviour: An exploratory study on github. In: Proceedings of the international workshop on crowd-based software development methods and technologies, pp 25–30

Jing Jiang received her B.S. and Ph.D. degrees in computer science from Peking University, China in 2007 and 2012, respectively. She is now an assistant professor in the State Key Laboratory of Software Development Environment of Beihang University, China. Her research interests include empirical software engineering, data mining, human factors and social aspects of software engineering.

David Lo received his PhD degree from the School of Computing, National University of Singapore in 2008. He is currently an assistant professor in the School of Information Systems, Singapore Management University. He has close to 10 years of experience in software engineering and data mining research and has more than 180 publications in these areas. He received the Lee Foundation Fellow for Research Excellence from the Singapore Management University in 2009. He has won a number of research awards including two ACM SIGSOFT distinguished paper award for his work on bug report management and bridging academia and industry. He has published in many top international conferences in software engineering, programming languages, data mining and databases, including ICSE, FSE, ASE, PLDI, KDD, WSDM, TKDE, ICDE, and VLDB. He has also served on the program committees of ICSE, ASE, KDD, VLDB, and many others. He is a steering committee member of the IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER) which is a merger of the two major conferences in software engineering, namely CSMR and WCRE. He is also serving as the general chair of ASE 2016. He is an experienced researcher in the emerging field of software analytics and has been invited to give keynote speeches and lectures on the topic in many venues, such as the 2010 Workshop on Mining Unstructured Data, the 2013 Genie Logiciel Empirique Workshop, the 2014 International Summer School on Leading Edge Software Engineering, and the 2014 Estonian Summer School in Computer and Systems Science.

Jiahuan He received his B.E. degree in software engineering from Beihang University, China in 2009. He is now a master candidate in software engineering of Beihang University. His research interests include empirical software engineering, data mining, and machine learning.

Xin Xia received his PhD degree from the College of Computer Science and Technology, Zhejiang University, China in 2014. He is currently a research assistant professor in the college of computer science and technology at Zhejiang University. His research interests include software analytic, empirical study, and mining software repository.

Pavneet Singh Kochhar Pavneet is a PhD candidate in School of Information Systems at Singapore Management University. He has previously done summer internship at Microsoft Research and an exchange programme at Carnegie Mellon University. His research interests involve data analytics for software engineering particularly focusing on software metrics, software testing and reliability. His work has been published in many international conferences such as ASE, ISSTA, SANER, ICST, MSR and QSIC. He has also served as an external reviewer for many conferences.

Li Zhang received her BS, MS, and PhD from the School of Computer Science and Engineering, Beihang University, China in 1989, 1992, and 1996, respectively. She is now a professor in the School of Computer Science and Engineering, Beihang University, China. She is a committee member of Software Engineering in China Computer Federation (CCF), committee member of education in CCF, and a committee member of computer applications in the Chinese Society of Astronautics (CSA). She is interested in software engineering, software architecture modeling, and system reliability analysis.