Coding together in a social network: collaboration among GitHub users

Dorota Celińska
University of Warsaw, Faculty of Economic Sciences
Warsaw, Poland
dcelinska@wne.uw.edu.pl

ABSTRACT

In this article we investigate developers involved in the creation of Open Source software to identify which characteristics favor innovation in the Open Source community. The results of the analysis show that higher reputation in the community improves the probability of gaining collaborators to a certain degree, but developers are also driven by reciprocity. This is consistent with the concept of gift economy. A significant network effect exists and emerges from standardization, showing that developers using the most popular programming languages in the service are likely to have more collaborators. Providing additional information (valid URL to developer's homepage) improves the chances of finding coworkers. The results can be generalized for the population of mature users of GitHub.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in collaborative and social computing**; • **Social and professional topics → User characteristics**; • **Software and its engineering → Open source model**;

KEYWORDS

Open Source, GitHub, forking, collaboration, innovations, reputation, gift economy, network externality, reciprocity

ACM Reference Format:
Dorota Celińska. 2018. Coding together in a social network: collaboration among GitHub users. In *SMSociety '18: International Conference on Social Media and Society, July 18–20, 2018, Copenhagen, Denmark*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3217804.3217895

1 INTRODUCTION

The Open Source software license allows the end users to study, modify, and distribute the publicly accessible source code to anyone and for any purpose. The creation of this kind of software usually relies on volunteer contributions and is associated with occurring communities, i.e., groups of people interested in, or working on, specific projects [41]. Communities form social networks, where innovations are created and distributed. By innovation, we mean "the process of commercialization of a newly developed or adopted product or practice" [17]. In particular, collaboration among developers which leads to the creation of a new product (new software, or improvements to previously existing one) can be viewed as a process which generates innovations.

The empirical research regarding Open Source software usually focuses on extracting the determinants of the Open Source license choice made on an enterprise or a project level [5, 13, 27, 32]. Additionally, a substantial literature revolves around explaining the incentives that motivate the individual developers to contribute to the Open Source projects [5, 22, 29, 44]. The theory of motivation regarding volunteer developers hardly differentiates between the individual and collective motives. The core focus of the analyses is the activity of a particular user, the network she is a part of is nearly never taken into account. The participation in Open Source have various dimensions – provision of freely accessible source code is only one of them. What makes developers more attractive to collaborate with seems to be lacking in the research. Additionally, most authors researching the topic of collaboration utilize data sets of small or medium size, which raises important issues about validity of results when trying to generalize them. Economists focus on the public good aspect of Open Source software. Although Open Source products are also subject to network externalities [5, 7, 38], it is surprisingly rarely mentioned in the relevant literature.

Socially connected computing is rooted in the study of corporate portals and groupware [18]. A great amount of research has been conducted on SourceForge, an Open Source repository that started in 1999 [16, 30, 37]. SourceForge, despite its popularity, lacks features to make social ties and keep up with other developers' updates [6, 31]. More recently implemented, GitHub is currently one of the largest repository hosting services related to the development of Open Source software, featuring elements of a social network service. As of Dec 31, 2016, GitHub has more than 15 million of registered users, and over 40 million of repositories. Not all repositories on GitHub are intended for collaboration – many are there just for personal or storage purposes [25]. Therefore, understanding what makes a developer attractive for collaboration (and how the network structures help in promoting them) is important and interesting.

We want to fill the noticed gaps. We will focus on developers whose projects have been forked, treating it as an action creating potential for further collaboration. In GitHub's jargon *forking* means creating a personal copy of the source code of another developer's repository. The user who forked a repository is allowed to suggest changes to the original repository via pull requests, which is not possible without forking. Studies on forking activity among GitHub developers are rare [11] and mostly aimed at software engineering

SMSociety '18, July 18–20, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-6334-1/18/07…$15.00
https://doi.org/10.1145/3217804.3217895

issues, rather than the analysis from the economic or social network point of view. As far as we know, the relationship between developers' characteristics and the receipt of pull requests have not been addressed in the research.

The aim of this article is to investigate how the characteristics (including network ones) of developers involved in the creation of Open Source software favor innovation in the Open Source community. Specifically, we analyze the volunteer developers. We will accomplish our objective by estimating the values of the parameters of logit model. The dependent variable will indicate whether or not the collaboration with the particular developer happened (and the innovation was generated). We will utilize a unique data set combined from three sources.

This article is organized as follows. In Section 2 we give a short review of the theoretical and empirical background, characterize the social networks described in this study, and present GitHub's fork&pull model of collaboration. In Section 3 we formulate the research hypotheses and introduce the empirical model. The data set and the results of the analysis are presented in Section 4. Section 5 provides discussion and implications of the results, in Section 6 we discuss the validity of the results and Section 7 concludes the research.

2 BACKGROUND AND RELATED WORK

Innovations in the Open Source community are created through collaboration among developers working in a distributed environment [26]. Open Source software constitutes a kind of contradiction from the perspective of classical economic analysis. The specific definition of ownership expressed in the license agreements and the common lack of direct remuneration for developers could discourage them from making effort, and therefore the source code would not be created. In fact, the behavior of developers is quite the opposite.

2.1 Developers' motivation

The current empirical research has identified three types of factors encouraging developers to write the publicly accessible source code [13, 42]: extrinsic motivation – factors originating from the external sources including the environment in which the developers work; intrinsic motivation – psychological factors driving developers; and internalized extrinsic motivation – factors that can be both external and psychological. The results of empirical analyses identified that determinants of making the decision whether or not to take part in creating Open Source code include the increase in subjective value of created software and satisfying one's needs by adding a missing functionality [22, 24, 32, 43]. Having fun while coding and a potential boost in career are also significant factors for developers [21, 22, 26, 30, 32]. According to Lerner and Tirole [32] developers decide to write publicly accessible source code because they expect it to enhance their employment opportunities. Taking part in the Open Source community and gaining visible popularity is treated as a kind of signal for the potential employers. The interest in developing Open Source software stems from a user's needs and the desire to fulfill those needs. Apart from economic factors, the developers of Open Source software are also driven by altruism [21, 43].

An abundance of "survival necessities", like computing power and disk space in the Open Source community, reveals this phenomenon's gift economy nature. Gift economy creates social structures and shapes the behavior of agents in case of excess instead of scarcity. In the case of Open Source software social status is related to "what you give away" and not to "what is under your control" [35, 40]. This leads to the situation in which reputation among developers' peers is the measure of competitive success. Developers' anticipated increase in reputation (popularity in the community) resulting from the number of fixed bugs encourages them to take part in writing the Open Source code [21, 22, 28, 30]. Because of gift economy, developers are not only interested in gaining reputation, but also expect the future gifts in return. The reciprocity as a motivational factor for developers was reported in various studies [3, 15, 30].

The diverse motivational components are not necessarily mutually exclusive and may co-exist within a developer. The main focus of this stream of literature is to understand how developers' motivations drive their participation or effort in Open Source projects which in turn affects the overall performance and effectiveness of developers and projects.

2.2 Social networks of GitHub

GitHub is a collaborative repository hosting service that includes social features bringing a new transparency to the development project [36]. The activity of its registered users forms several kinds of social networks. The most intuitive one is the network of collaboration among developers within the project repositories. The existence of this network stems directly from the Open Source licenses' statements, that enable modifications of the publicly accessible source code. Because of the special model of collaboration in GitHub repository hosting service, this network is embedded in the observable network of users that are granted permission to contribute to the projects, namely its members. Collaboration among GitHub users can be seen in at least two ways. Firstly, one can analyze bipartite graphs of developers and their particular repositories they contribute to. Secondly, as suggested by Lima et al. [33], one can project the mentioned graph onto the set of users – this way, developers who contribute to at least one common repository are connected to each other. Moreover, we utilize the third way of mapping the relationship of collaboration: the connection between the initial owner of the project repository and the members of the repository (however, the members themselves are not connected). This way we preserve the hierarchical nature of the most popular model of collaboration in GitHub.

The second type of social network in GitHub is the network of followers – users who agreed to be sent notifications about one's activity on GitHub. The network of followers may serve as a proxy of influence and reputation process in the service. As Goggins and Petakovic [18] state: following a developer gives them a degree of influence. The more followers a developer has, the larger the group potentially interested in their work, creating a potential for even greater influence. Even if not capable of contributing to the projects of "globally" popular developers, users of GitHub are interested in their work. This explains enormous numbers of followers of celebrities, like GitHub's staff. On the other hand, following less

popular developers improves the efficiency of matching possible coworkers. This is closely related to the spillover effect. Knowledge spillovers are reported by Fershtman and Gandall [16] as correlated with the structure of the social networks, because contributors who work on several projects are likely to exchange information and knowledge. While following a person, one is alerted to their activity on the service: the creation of new repositories, submitted commits, issues and pull requests, along with starring (giving a distinction to) another developer's project repository. The process of following should considerably decrease the cost of acquiring information about the possible coworkers and the projects one would like to contribute to, because developers are quickly given the notification instead of browsing the service on their own. What is more, the analysis of 199 GitHub's most followed users and their followers by Blincoe et al. [4] showed that popular users influence their followers by guiding them to new projects.

The network of watchers is similar to that of followers, though of different type of information is provided. A watcher is a follower of the particular repository. While notifications stemming from following a developer supply quite a diverse range of information, the alerts for watchers are confined to the activity within one repository (mostly issues, bug fixes, and the collaborations within project). Watchers have probably been already interested in the development of a particular project, therefore we do not consider this network in our study.

2.3 Github's fork&pull model of contribution

Every registered user of GitHub has their own site that integrates social media functionality directly with code management tools [34]. Each individual user's profile contains public information about their biographical characteristics (the date of joining the service and the optional description of location, employer, personal site and e-mail address), the list of public project repositories, the number of people following the user, as well as the number of people that the user follows. Everyone, even unregistered users of the service, may browse the profiles and download public project repositories.

The contribution to the project may occur in at least two ways [6]: by sending via e-mail the patches to the author of the original project; or by submitting the pull requests – proposed changes to the projects that may be accepted or declined by the maintainers of the project. The submission of pull request requires the prior registration in GitHub. The developer willing to commit to the original repository (and thus make it an upstream repository commit) has to fork the existing project repository. Forking means creating developer's own copy of the source code of other developer. As a result the user (according to the GitHub's jargon now known as a member) is granted the access to the complete project. The access also includes all branches that were created before the project had been forked. The members may introduce their commits (proposed bug fixes and changes to the source code) locally, but are also allowed to synchronize the state of their repositories with the state of the original project. A pull request is a feature of GitHub where the developer sends the suggested modifications to the maintainer of the original project repository. If the maintainer of the original project repository accepts the changes introduced in the pull request, both repositories are merged and the history of submitted commits by the individual users is preserved. Forks of repositories are potential new projects and they may operate independently from the rest of the original projects. However, it is difficult to determine if the exact fork will or will not contribute back to the base project [24].

2.4 Reputation and reciprocity in GitHub

Reputation seems to be the only motivational factor known from the research on Open Source, which is investigated in the literature concerning GitHub. Perceived reputation increases the declared likelihood and interest in future contribution to Open Source software [45]. In GitHub, proxies of reputation and "social status" of the developers emerge from the following network and the starring and watching networks. In the literature popular and influential users of GitHub are known as rockstars. A rockstar is characterized by having a large number of followers who are interested in their coding activity and what projects they follow or work on [2, 14, 23]. The number of followers is treated as a signal of popularity/status in the community. Popular users are loosely interconnected among them – usually they have more connections with unpopular users than with other popular ones [1, 33].

Users become popular as they write more code and monitor more projects. However, gaining popularity is non-linear; while low popularity levels can be attained with a little effort, achieving higher levels depends on more effort. What is more, popularity does not result from development alone, e.g., Lima et al. [33] report that there is a weak relation between the number of followers of a user and her contributions, which may suggest that even a high level of activity may not directly attract a large number of followers. The popularity of the user may depend on different factors, e.g., participating in different projects and discussions. Rockstars play an important role on the project dissemination and attractiveness. In particular, when a rockstar increases her activity on a project, it attracts more followers to participate (e.g. opening issues or commenting in their threads) on the same project [4, 31].

According to the literature a proxy for both reciprocity and altruism can be found in one's attitude towards providing and receiving help. In recent GitHub Open Source survey [45] 72% of respondents declared to ever having provided help for another person in an Open Source project and 71% to receiving any kind of help from the community. The provision and receiving of help are not independent – the correlation between those two variables is statistically significant (Kendall coefficient = 0.84, p-value = 0.000). The students receive and provide help significantly more frequently than the employed ones (proportion tests, p-values = 0.000). There are no significant differences in the shares of respondents providing and receiving help among the employed ones (proportion tests, p-value for receiving help = 0.28 and for providing help = 1.00).

3 METHODOLOGY

We consider a group of agents (developers) $V = \{1, \ldots, N\}$, who are members of a social network represented by a directed graph $\mathcal{G}(V, E)$, where the set of edges E represents connections between agents. Self-loops are not allowed in this graph. Each agent (a node in the graph) is described with a set of characteristics, such as number of repositories hosted on GitHub, number of stars (distinctions)

given to their projects, number of followers, number of followed developers, number of watchers, etc.

We consider six types of social networks. We represent the relation of forking one's project repository (and thus creating an innovation by collaboration with the owner of the repository) with directed graph $\mathcal{G}_m$, which we call the members graph. The edges of this graph show the hierarchical structure of forking: there is an edge in $\mathcal{G}_m$ between A and B iff A is a member of a project repository owned by B. A closely related graph is the $\mathcal{G}_{pr}$, pull request graph, in which there is an edge between A and B iff A submitted a pull request to B (A must be B's repository member). Directed graph $\mathcal{G}_i$ captures the events related to issues, and there is an edge between A and B in directed graph $\mathcal{G}_c$ iff A writes a comment for B. The followers graph $\mathcal{G}_f$ represents the following relations among developers. There is an edge in $\mathcal{G}_f$ between A and B if and only if A follows B. The last considered graph is the starring graph $\mathcal{G}_s$ in which an edge corresponds to giving a distinction to a developer's repository.

3.1 Research hypotheses and model design

We will verify following hypotheses:

Hypothesis 1. *The reputation proxies positively affect the probability that others would like to collaborate with a particular developer. However, this impact is nonlinear.*

Hypothesis 2. *The reciprocity proxies positively affect the probability that one would get the coworkers (one of the motivational factors for developers is reciprocity).*

Hypothesis 3. *There exists a significant network effect emerging from standardization: the users of the most popular programming languages tend to be more likely to have collaborators.*

We utilize two binary logit models. Logit models form the family where the dependent variable is categorical. The binary logit model is used to estimate the probability of a binary response based on one or more independent variables [12]. In our case, in the first model we estimate the probability of having a repository forked (we will call it *forkability* as suggested in [11]), and in the second one we estimate the probability of obtaining a pull request, under the condition of having a repository forked (similarly, we will call it *pullrequestability*). The independent variables are same in both models, we divide them into five sets:

- **Reputation**: the number of developer's *followers*, *stars obtained* by a developer (the sum of stars given to developer's repositories), the *evc_following* (eigenvector centrality of developer in $\mathcal{G}_f$);
- **Reciprocity**: the number of developers *followed* by a developer, the number of repositories *forked* from others, *stars given* to other developers' repositories, and *commits* pushed to repositories owned by other developers;
- **Communication**: the number of *comments* written in developers' own and in others' repositories and the number of *issues* opened in own and others' repositories;
- **Standardization**: a set of dummies indicating whether developer owns a project written in one of the 14 most popular *languages* in the service – the languages were chosen initially basing their position in [9] and if they were used by at least 5% of users in the sample, the number of languages used in developer's repositories, the fraction of specific usage intended languages in all of the languages used by developer (specialization);
- **Information**: a set of dummies indicating whether developer provides a valid *e-mail* address or a valid URL to their personal *site*, the number of *repositories*, the *year* of developer's registration, the number of *commits* in the base repositories (the ones that were not forked) and in forked repositories.

The variables in the *communication* set provide ambiguous information. On one hand, they are related to the *reciprocity* set, because e.g., by opening issues one can report bugs or propose features. On the other hand, excessive number of issues and comments may also indicate the developers with low experience, who are not capable of finding solutions on their own. We are not able to distinguish among the types of the issues and comments written, mostly because of the natural missing data patterns in the database – GitHub API does not provide the bodies of the entries from 2013 to 2014. That is why we decided to enclose those variables in a separate set.

We suppose that the continuous variables may have nonlinear impact on the probabilities. That is why we also introduce the squares of those variables.

We introduce both the degree centralities and eigenvector centralities as the proxies for reputation. We decided not to utilize the Kleinberg centralities for the simplicity of the interpretation. Since the structures of the networks are not independent [6], we assume that the characteristics of a node in other networks can be used as regressors.

4 RESULTS

4.1 Data set

GitHub data is huge and the service is continuously evolving, therefore the complete download of the data is impossible [19, 25]. To minimize the number of missing observations, we use a data set combined from three sources: GHTorrent project [19], GitHub Archive project [20], and our own database obtained by web-scraping GitHub in 2016. Web-scraping means automatic extraction of information from websites by web crawlers – Internet bots that systematically browse the World Wide Web. Scraping is focused on the transformation of the unstructured data on the web, typically in HTML format, into structured data that can be stored and analyzed, e.g., in a spreadsheet. The data about GitHub's registered users is publicly available, but very distributed. To combine the data, we used a set of heuristics. GitHub Archive data in JSON format contained information about the service unique id, the same as data obtained by our web-scraping. Data coming from GHTorrent lacked this information, but we were able to merge records using users' logins together with the date of registration. The events, which we used to create networks, were merged by the standardized timestamps. We consider user accounts sharing the gravatar, or using the same login at different times, to be the same user, and thus such accounts are merged into a single observation using Find-Union algorithm [10]. The data we utilize and collect is publicly available. Revealing user's personal data, e.g., e-mail address is not mandatory, the user decides – that is why we suppose

that the information we collect does not violate anyone's consent. No special ethical consideration applies here.

The population of this research consists of users of GitHub registered in 2007-2014, which means 10,361,315 entities. We limited the span of registrations to make sure the developers had a chance to gain popularity, learned how to use the service and had enough time to start collaborating with others. Similarly to Lima et al. [33], and Kalliamvakou et al. [25] our sample is smaller: it consists only of registered users that are active (did not delete their profiles) and have at least one public repository. We also disregard the "organization" accounts due to the non-random missing data patterns connected with those profiles. The resulting data set contains information about the activity of 3,915,138 users.

The continuous variables are not normally distributed, as could be expected based on the stylized facts provided in other studies [1, 2, 6, 33]. Networks emerging from similarity and popularity (reputation) mechanisms tend to show power-law scaling behavior. However, the outliers could threaten the results of our analysis, that is why we decided to remove the top one % of observations for every continuous variable. In resulting sample 25.9 % of the developers have at least one repository forked, whereas 57.6 % of developers forked someone's repository. The average user from our sample has 6 public repositories, 2 followers, follows 2 users, and obtained 2 stars. The median and mean values in the sample of developers, whose repositories have been forked are usually higher than in the full sample (Table 1).

Five most popular programming languages in the full sample are: JavaScript (40 % of users), CSS (31 % of users), HTML (25 % of users), Java (24 % of users), and shell (22 % of users). In samples of users, who have at least one repository forked the hierarchy of popularity is mostly the same: Python (40 % of users) is slightly more popular than Java (39 % of users). Developers, who have at least one repository forked, no matter if they received pull request or not use a greater number of different programming languages compared to the whole population (Table 2), this observation is consistent with the results reported by Badashian and Stroulia [2], where popular users extert their influence in more than one programming language. The popularity of the programming languages in sample of developers who received pull request seems to not vary visibly from the corresponding popularity in the sample of developers whose repositories have been forked.

We will analyze the impact of centralities on the collaboration with a given developer, which is why we investigate to what extent the influential nodes found for a given network are also influential in other networks. We assume that GitHub users can be ranked with their influence: we compute the degree centralities F_D for all of the networks considered in this study; then we obtain a subset of the vertices characterized by centrality degree value higher or equal 90, 95, 99 centile of the metrices' distributions. We also distinguish between in-degree centrality F_{D-in} (the in-degree of the node) and out-degree centrality F_{D-out} (the out-degree of the node). We start with treating the networks as the subnetworks of a multilayer network, and check what percent of nodes is at the same time influential in a given number of subnetworks (tab.3). Because analyzing every possible combination of being influential or not in the subnetworks would result in analyzing 63 entries (we are not interested in nodes not being influential at all), we focus on the analysis of the nodes shared among pairs of networks (Table 4).

Usually there are more nodes with high out-degree that in-degree, the difference occurs among the nodes characterized by the top percent of in- and out-degrees. The difference lies in the source of high in- or out-degrees; usually it is easier to start multiple relations with other nodes in the network (to have high out-degree) than to attract many vertices at the same time (to have high in-degree). However, maintaining multiple links requires increasing costs, e.g. following many developers or starring many repositories will end with the feed flooded with notifications, leading to informational chaos – just the opposite of the desired output it was aimed at. On the other hand, users characterized by top-percent in-degrees are not affected by a similar cost. They have already gathered noticeable attention within the network (we do not discuss here why), and probably will gather further links, due to power-law scaling behavior observed in a network.

Analyzing data gathered in Table 3 and 4 we notice that about half of the influential users are influential in only one network, no matter if we discuss results for in- or out-degrees. However, apart from users belonging to 99th centile of F_{D-out} about one third of users are influential in at least three networks at the same time. This result suggests that popularity may be a more complex research topic, and probably the position of the node in the networks should be analyzed from a broader multilayer perspective. Having similar influencers in many networks affects also the possibilities for network effects and gives basis for the strategical choose of links, e.g., creating a link in the following $\mathcal{G}_f$ network to a user which is influential at the same time in the collaboration related networks reduces cost of obtaining information.

4.2 Estimation results

The results of the analysis in the form of the values of coefficients of logit model are presented in Table 5. The functional form of our model is correct (linktest's p-value > 0.05).

The coefficients for the independent variables from the *reputation* set were always statistically significant. The significance of the squares of the variables supports Hypothesis 1 about the non-linear impact of reputation proxies on the probability of collaborating with a developer. Developers with excellent programming skills are likely to be highly rewarded in the community, but their projects may be characterized by such high entrance costs and the degree of specialization that they discourage other developers from collaboration. Thus the reputation proxies generally evince diminishing returns to scale. The only exception is the case of eigenvector centrality of the node in the network of following, which shows increasing returns to scale. This result suggests existence of a special kind of a network effect among GitHub developers – a developer's position in the network does not depend only on the number but also on the quality of the connections they have. However, while examining the probability of having a pull request back if having a repository forked, all of the reputation proxies show increasing returns to scale. This is also consistent with intuition: developers with higher reputation in the community may have less coworkers, but if someone decides to fork their repository, they do want to collaborate, instead of for example just making a personal backup.

Table 1: Descriptive characteristics of the sample

		Forkability					*Pullrequestability*				
Set	Variable	Min	Max	Median	Mean	% of values>0	Min	Max	Median	Mean	% of values>0
Reputation	followers	0	35	0	1.448	34.947	0	35	1	3.675	64.502
	stars_obtained	0	140	0	1.877	31.816	0	140	1	6.074	60.763
	evc_following	0	2000	0	50.00	28.195	0	2000	1	105.2	51.179
Reciprocity	following	0	40	0	1.430	28.928	0	40	1	3.200	50.406
	stars_given	0	128164	0	822.9	47.044	0	358	2	17.59	68.089
	forked_repos	0	80	1	2.386	57.601	0	80	2	5.117	73.245
	commits_others	0	4436	0	14.88	37.826	0	4436	5	40.70	70.240
Communication	comments_self	0	130	0	0.562	7.164	0	130	0	1.999	24.228
	issues_self	0	110	0	0.571	7.342	0	110	0	1.666	19.541
	comments_others	0	800	0	5.036	26.982	0	799	0	13.51	49.186
	issues_others	0	200	0	1.832	24.700	0	200	0	4.498	44.304
Standardization	langcount	0	145	2	3.634	76.523	0	144	5	7.026	93.352
	fraction_functional	0	1	0	0.008	4.888	0	1	0	0.014	11.167
	fraction_web	0	1	0	0.049	25.391	0	1	0	0.074	48.821
	fraction_scientific	0	1	0	0.022	7.676	0	1	0	0.027	14.770
Information	repositories	1	80	2	5.906	100	1	80	8	12.96	100
	commits_self_base	0	2200	2	30.48	56.615	0	2200	26	86.13	88.875
	commits_selffork	0	750	0	4.398	23.604	0	750	0	13.17	48.599

Source: Own elaboration

Table 2: Usage of the programming languages among GitHub users

Language	All developers	Developers having at least one repository forked	Developers who received at least one pull request
JavaScript	40%	63%	65%
Java	24%	39%	38%
Ruby	16%	29%	32%
Python	21%	40%	41%
PHP	16%	29%	30%
CSS	31%	55%	59%
HTML	25%	48%	54%
C++	14%	28%	27%
C	15%	29%	29%
Objective C	9%	17%	16%
C#	9.1%	15%	15%
Shell	22%	44%	46%
Perl	6.4%	14%	14%
CoffeeScript	5.8%	12%	14%

Source: Own elaboration

Table 3: Degree centrality – percent of nodes which are influential in simultaneous number of networks with respect to the size of samples

Subset	Overall	% of unique nodes	Percent of nodes common in subsets: 6	≥ 5	≥ 4	≥ 3	≥ 2
90th centile F_{D-in}	523,803	52.9	8.5	16.8	24.1	33.1	47.1
90th centile F_{D-out}	2,452,024	47.4	4.2	11.0	19.4	30.6	52.6
95th centile F_{D-in}	269,839	48.8	8.9	18.6	26.8	36.2	51.2
95th centile F_{D-out}	1,114,081	50.6	3.8	10.1	18.3	28.5	49.4
99th centile F_{D-in}	87,043	41.0	8.3	20.1	30.4	41.0	59.0
99th centile F_{D-out}	121,166	68.2	1.1	3.0	6.8	13.6	31.8

Source: Own elaboration

Table 4: Degree centrality – percent of influential nodes shared among pairs of networks

pair	Percent of common influential nodes					
	90th centile		95th centile		99th centile	
	F_{D-in}	F_{D-out}	F_{D-in}	F_{D-out}	F_{D-in}	F_{D-out}
$\mathcal{G}_c \times \mathcal{G}_f$	15.1	16.7	16.4	16.2	18.6	7.6
$\mathcal{G}_c \times \mathcal{G}_s$	46.6	31.2	47.5	51.0	44.7	8.6
$\mathcal{G}_m \times \mathcal{G}_c$	39.7	30.7	39.4	27.9	39.8	12.5
$\mathcal{G}_m \times \mathcal{G}_f$	28.8	15.3	29.3	15.8	26.7	11.9
$\mathcal{G}_m \times \mathcal{G}_s$	52.9	35.3	48.6	32.1	42.6	12.8
$\mathcal{G}_i \times \mathcal{G}_c$	50.0	59.7	57.0	61.6	67.4	51.1
$\mathcal{G}_i \times \mathcal{G}_f$	21.7	15.9	21.1	14.6	19.9	8.1
$\mathcal{G}_i \times \mathcal{G}_m$	47.0	27.5	48.7	26.6	46.7	12.2
$\mathcal{G}_i \times \mathcal{G}_s$	45.2	29.0	47.7	25.2	43.3	10.0
$\mathcal{G}_{pr} \times \mathcal{G}_c$	48.1	32.6	54.4	41.5	55.8	16.7
$\mathcal{G}_{pr} \times \mathcal{G}_f$	18.7	20.8	18.9	17.7	19.6	8.3
$\mathcal{G}_{pr} \times \mathcal{G}_m$	45.8	23.0	44.7	25.1	40.5	15.9
$\mathcal{G}_{pr} \times \mathcal{G}_i$	44.7	29.4	48.4	33.5	48.8	43.7
$\mathcal{G}_{pr} \times \mathcal{G}_s$	38.8	22.5	41.4	19.9	40.2	7.5
$\mathcal{G}_s \times \mathcal{G}_f$	23.4	24.3	24.4	22.4	26.8	19.6

Source: Own elaboration

Those results are consistent with the result obtained in [1, 6, 33] about the power-law scaling behavior of the $\mathcal{G}_f$. The results regarding reputation are similar to those described in the literature [22, 30, 32].

The results obtained for the *reciprocity* set of variables support Hypothesis 2. Developers are driven by reciprocity as reported in [3, 15, 30], but only up to a certain degree. Exaggeration while showing interest in others' work may lead to perception of a diminishing value of that interest. Only the distinctions given to other developers always show increasing returns to scale. We can also give similar interpretation to the values of coefficients of the *communication* set of variables. Developers visibly dependent on the help from community, probably with lower reputation and lower programming skills, may discourage from collaboration with them. Those results are also valid for the probability of receiving a pull request (*pullrequestability*).

The coefficients of the programming language related variables were mostly statistically significant. Having a project repository written in one of the most popular languages in the service usually increases the probability of finding new collaborators. The exceptions are Ruby (insignificant value of coefficient) and HTML, probably because of the nature of this language – HTML code is usually about the exact webpage, not about the collaboration and providing a broader, sustainable solution. The results obtained in the case of forkability generally support Hypothesis 3 that the Open Source software is characterized by a network effect derived from specialization. This kind of externality can be also seen as the *lock-in effect*. Specialization of the developer also improves their chances of finding new potential collaborators (values of the coefficients for *fraction* variables greater than zero). However, while examining the network effect in the case of *pullrequestability*, the usage of the popular languages may even decrease the chance of having a pull request back. The only languages always increasing the probability are CSS and CoffeeScript. Repositories in popular languages attract new developers, creating potential for later collaboration, but for

Table 5: Values of coefficients of logit models

		Forkability	*Pullrequestability*
Set	Variable	Coef.	Coef.
Reputation	followers^2	0.1257 ***	-0.0080 ***
	followers^2	-0.0040 ***	0.0006 ***
	stars_obtained	0.1423 ***	-0.0098 ***
	stars_obtained^2	-0.0010 ***	0.0001 ***
	evc_following	-0.0006 ***	-8.717e-05 ***
	evc_following^2	2.11e-07 ***	5.552e-08 ***
Reciprocity	following	-0.0007	0.0039 ***
	following^2	-0.0002 ***	-0.0001 ***
	stars_given	-0.0026 ***	-0.0006 ***
	stars_given^2	7.036e-06 ***	1.122e-06 ***
	forked_repos	-0.0512 ***	0.0255 ***
	forked_repos^2	0.0007 ***	-0.0005 ***
	commits_others	0.0019 ***	0.0005 ***
	commits_others^2	-5.862e-07 ***	-1.416e-07 ***
Communication	comments_self	0.0995 ***	0.0755 ***
	comments_self^2	-0.0009 ***	-0.0006 ***
	issues_self	0.0042 ***	0.0087 ***
	issues_self^2	-0.0003 ***	-0.0002 ***
	comments_others	0.0023 ***	0.0026 ***
	comments_others^2	-4.019e-06 ***	-3.891e-06 ***
	issues_others	0.0059 ***	0.0033 ***
	issues_others^2	-4.95e-05 ***	-3.071e-05 ***
Standardization	javascript	0.0623 ***	-0.1133 ***
	java	0.1523 ***	-0.1653 ***
	ruby	-0.0011	0.0512 ***
	python	0.1283 ***	-0.0380 ***
	php	0.1040 ***	-0.0570 ***
	css	0.1162 ***	0.0258 ***
	html	-0.0738 ***	0.0977 ***
	c++	0.0825 ***	-0.0779 ***
	c	0.0682 ***	-0.1248 ***
	objectivec	0.0255 ***	-0.0887 ***
	c#	0.1141 ***	-0.0344 ***
	shell	0.1084 ***	-0.0141 ***
	perl	0.0136 ***	-0.0279 ***
	coffeescript	0.0027	0.0520 ***
	langcount	-0.0109 ***	0.0073 ***
	fraction_functional	0.0008	-0.1464 ***
	fraction_web	0.5122 ***	0.3812 ***
	fraction_scientific	0.2175 ***	-0.4352 ***
Information	e-mail	0.0247 ***	-0.0375 ***
	site	0.0342 ***	-0.0012
	repositories	0.0855 ***	-0.0028 ***
	repositories^2	-0.0012 ***	9.027e-05 ***
	year2009	-0.0544 ***	0.0637 ***
	year2010	-0.0268 **	0.0919 ***
	year2011	0.0096	0.1298 ***
	year2012	0.1146 ***	0.1938 ***
	year2013	0.1368 ***	0.2843 ***
	year2014	0.2420 ***	0.4743 ***
	commits_self_base	0.0021 ***	0.0009 ***
	commits_self_base^2	-1.52e-06 ***	-5.731e-07 ***
	commits_self_fork	0.0102 ***	0.0058 ***
	commits_self_fork^2	-1.675e-05 ***	-9.103e-06 ***
	intercept	-1.7284 ***	-0.2518 ***

Source: Own elaboration

* denotes variable significant at significance level 0.1

** denotes variable significant at significance level 0.05

*** denotes variables significant at significance level 0.01

many of those developers the collaboration with the initial owner may not be the primary intention of forking. They may for example be interested just in adjusting the existing source code to their specific case and personal needs, like in the original definition of the fork. Our results are consistent with Rastogi and Nagappan [39], who divide forks into three kinds: contributing forks, which are the source of pull requests for the original project, independently developed forks, which develop new, deviating from the original versions of project, and inactive forks, which seems to be just sort of backup copies.

People providing valid URLs to their personal sites reduce the cost of obtaining information about them. It also helps in forming one's impression in the service which corresponds with the suggestions of Marlow et. al. [34]. The negative sign of the value of the coefficient of e-mail (for model for probability of receiving a pull requests) may be explained by the usual process of gaining e-mail addresses in software development based on git control system. The typical commit usually includes commiter's e-mail address, so it is quite easy to obtain it, and publicly available one is not necessarily important for other developers. What is more, very active users of GitHub may even hide their e-mails, e.g., because they do not want to receive spam e-mails.

5 DISCUSSION AND IMPLICATIONS

This study focuses on determining the factors enhancing one's probability of gaining collaborators in GitHub. In particular, we investigate the characteristics of the graph nodes. A competitive approach basing on the characteristics of dyadic data sets would provide further insights into the determinants of collaboration among developers. We suppose that the set of variables utilized in this study could be augmented: further studies would include e.g. the gender aspects and the spatial analysis of the collaboration in the service.

To practitioners, our study provides insight how they can make their projects more popular and active, and what characteristics should they pay attention to, while seeking new collaborators. Because the user's profile on GitHub can be treated as their resume, the results of this analysis may also be interesting for the recruitment purposes, especially for the recruiters searching for developers showing skills in team working.

6 THREATS TO VALIDITY

Simultaneity. We are aware of the possible endogeneity issues related to the functional form of our model. Endogeneity seems to be quite popular but little discussed problem in socio-economic research. As it was shown in [6] the structures of networks considered in our study are not independent. The sequence rule mining [8] provides some information, what patterns of creating multiple relationships among developers are more frequent than others. The structure of a network should not be analyzed independently from the other ones. Despite this it seems intuitive that users characterized by a lot of followers may at the same time have many coworkers who they acquired because of the excellent programming skills, and that they have many followers because of those programming skills. We are not able to state the "causality" here, even it was showed that following and then starting collaboration seems to be more frequent scenario among GitHub users. To address the endogeneity issue (or at least minimize its impact on the values of the marginal effects) in theory we could perform instrumental variables estimation. However, there are few issues, which discouraged us from doing so.

- **Choosing the correct instruments**
  The first and the most important issue is that we are not able to choose correct instruments. The instruments must be correlated with the endogenous explanatory variable and cannot be correlated with the error term in the explanatory equation, both conditional on the value of other covariates. However, as we have shown, most of the users' activities within service is not independent and we have limited external information about the users. Based on the available data we are not able to find variables which would not suffer the same problem as the original predicting variables. We also believe that there are no reasons gender or geographical location (which could be imputed, bearing in mind the low quality of such variables) could be good instruments. Of course, one may suggest mining other services; taking into account the size of the sample we would not be able to make sure we avoided most of the false positives. We cannot hope that all of the users used unique and "stable" account names.
- **Complexity of the functional form**
  Since the structures of all of the networks considered in our study are correlated, the endogeneity is a more complex problem in our model than the usual *one problematic variable case*. We should introduce many new instruments and additional equations for every suspectible variable, leading to unnecessary complex multi equation model.
- **Drawbacks of Linear Probability Model**
  Finally, assuming we managed to find instruments, we would estimate the Linear Probability Model, which even if computationally achievable, bears some well-known problems. If we do not place restrictions on the values of coefficients, we have no guarantee that the resulting probabilities would not be implied outside of the unit interval [0,1]. This does not occur while using logit or probit models. The error term in Linear Probability Model is also always heteroscedastic – one should obtain the Feasible Generalized Least Squares estimator basing on the correct values of probability. The distributions of the variables are skewed, even after trimming one percent of observations we assume that a significant number of observations would be excluded during this procedure leading to a great information loss.

Aware of plausible simultaneity bias, we decided not to interpret the exact odds ratios or marginal effects. We discussed the signs of the values of coefficient and the statistical significance, which should be less affected. Our results do not contradict the results known from the literature, therefore we suppose that in the given framework they do not suffer from the discussed issue.

Lack of qualitative information. Data mining can unintentionally be misused, and can then produce results which appear to be significant, however, they may stem from the actual sample and cannot be reproduced later. Therefore we are cautious with the results which we obtained. We aim to find the hidden patterns behind the

creation of the links among users without asking those users what are the actual reasons for establishing links. This limitation is hard to mitigate. In the opinion of the author, in recent years surveying developers was significantly overused and led to the reluctance of the respondents. Furthermore, even if the survey was carried out, the standard concerns related to survey data would apply, regarding the sampling techniques, generalizability of the results, and even the validity of the responses, which would be declarative ones.

Generalizability of the results. Due to the power-law scaling behavior characterizing networks emerging from GitHub as shown by [6, 33], the results are representative for the population of active users of GitHub in the case of forkability. However, even if we obtained quite intuitively explainable results for *pullrequestability*, we suggest their cautious generalization. We expect the network of pull requests to contain missing observations, which are impossible to impute. We may only assume that those missing observations do not distort the actual network, because of the fractal nature of the networks in GitHub. Another limitation of the current study is that the utilized data set contains mostly information about mature users of the service. We lack the comparison with the possible "newcomers", i.e., users who have recently registered.

Sampling issues. We are fully aware that our data set suffers a time bias. We are not able to collect data about the whole repository hosting service in a preferably short time. The data utilized in this study is a snapshot of the service, the exact networks may differ. However, in most cases we do not find this observation a threat to the obtained results; rare random missing observations should not impact the correlation structure, which was the topic of our interest. The only exception is the case of pull request network as discussed earlier.

7 CONCLUSIONS

The analysis of the relationships among the Open Source software developers is a rare subject of the relevant literature. In this article we have shown that higher reputation in the community improves up to a certain degree the probability of gaining collaborators, but developers are also driven by reciprocity. This is consistent with the gift economy concept. There exists also a statistically significant network effect emerging from the standardization. Providing additional contact information also improves the chance of having collaborators.

We provided a broader analysis of the tails of the distributions, that indicate the possible influencers in the networks. We found that quite a high share of nodes tend to play important roles in at least two networks at the same time. This percentage is higher in the subsets of nodes related to in-degree than out-degree. It can be easily explained: in the directed edges, those nodes are the ending ones, so the cost of these connections usually lies on the side of the starting nodes. The diversity in the out-degree is higher, because the cost of link maintenance relies on the starting node. Analysis to what extent the influential nodes are shared among networks is related to both the network effect issues and the robustness of the network.

The obtained results provide empirical support for the existence of network effects of various kinds in the community of GitHub Open Source developers. However, the traditional approach to network effects, in which an agent only takes into account the number of other agents performing an action, is insufficient to describe the phenomena observed in this study. One can easily explain the signs of the coefficient for the popular programming languages while interpreting the forkability, but the traditional way of understanding network effects struggles to explain the opposite signs in the case of probability of receiving a pull request back. That is because the traditional understanding does not distinguish between the impact of the *potential* and the *actual* network one is affected by. Forkability is related to the potential network of collaborators, but those developers will not necessarily form the actual one. The traditional understanding of the network effect also neglects the node position in the network, focusing only on the number of nodes. Using this approach, we are not able to observe more complex interactions which take place in the network. Utilizing the traditional approach, we would be able only to point that there is a network effect in the number of users, because of the positive values of the coefficients for registration year after 2011 – submitting the source code to GitHub gains value, the more users are registered to the service. The traditional approach however leads to counterintuitive reasoning. The developer while deciding whether or not to perform action takes into account the number of other developers already performing this action, not caring that most of those considered developers will never obtain information about their activity and will not interact with them. One has to examine the actual network of information dissemination, rather than the potential one, to capture the real network effect that influences the behavior of the entities. What is more, the position of the node in the network and the quality of the connections also strengthen the network effects, which cannot be observed in the traditional approach.

This analysis differs from the previous research in various dimensions. Firstly, we propose a different specification of the collaboration network, preserving its underlying hierarchical structure. We also limit the population of the users, to assure that they had enough time to gain popularity and collaborate with other developers. Furthermore, we apply statistical tests and combine the social network analysis with econometric model to quantitatively describe the phenomenon. The obtained results can be generalized for the population of mature users of GitHub.

8 ACKNOWLEDGMENTS

The author is very grateful to the anonymous referees for their careful reading of an earlier version of this work. Many parts of the paper have been greatly improved as a result of their insightful and constructive comments. This work is supported by the National Science Centre, Poland, grant DEC-2016/21/N/HS4/02100.

REFERENCES

[1] Mohammad Y. Allaho and Wang-Chien Lee. 2013. Analyzing the social ties and structure of contributors in Open Source Software community. In *2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*. 56–60.

[2] Ali Sajedi Badashian and Eleni Stroulia. 2016. Measuring User Influence in GitHub: The Million Follower Fallacy. In *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering (CSI-SE '16)*. 15–21.

[3] Magnus Bergquist and Jan Ljungberg. 2001. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal* 11, 4 (2001), 305–320.

[4] Kelly Blincoe, Jyoti Sheoran, Sean P. Goggins, Eva Petakovic, and Daniela Damian. 2016. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information & Software Technology* 70 (2016), 30–39.

[5] Andrea Bonaccorsi and Cristina Rossi. 2003. Why Open Source software can succeed. *Research Policy* 32, 7 (2003), 1243–1258.

[6] Dorota Celińska. 2016. Information and influence in social network of Open Source community. In *9th Annual Conference of the EuroMed Academy of Business*. 485–495.

[7] Dorota Celińska. 2016. Why do users choose Open Source software? Analysis of the network effect. *Informatyka Ekonomiczna. Business Informatics* 1, 39 (2016), 9–22.

[8] Dorota Celińska. 2018. Temporal relationships in GitHub. (2018).

[9] Dorota Celińska and Eryk Kopczyński. 2017. Programming Languages in GitHub: A Visualization in Hyperbolic Plane. (2017), 727–728.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald R. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. The MIT Press.

[11] Valerio Cosentino, Javier L. CÃąnovas Izquierdo, and Jordi Cabot. 2017. A Systematic Mapping Study of Software Development With GitHub. *IEEE Access* 5 (2017), 7173–7192.

[12] David R. Cox. 1958. The regression analysis of binary sequences (with discussion). *Journal of the Royal Statistical Society. Series B (Methodological)* 20 (1958), 215–242.

[13] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. 2008. Free/Libre Open-source Software Development: What We Know and What We Do Not Know. *Comput. Surveys* 44, 2 (2008), 7:1–7:35.

[14] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. 1277–1286.

[15] Paul A. David and Joseph S. Shapiro. 2008. Community-baased production of open-source software: What do we know about the developers who participate? *Information Economics and Policy* 1, 20 (2008), 364–398.

[16] Chaim Fershtman and Neil Gandal. 2011. Direct and indirect knowledge spillovers: the "social network" of open-source projects. *The RAND Journal of Economics* 42, 1 (2011), 70–91.

[17] Chris Freeman and Luc Soete. 1997. *The Economics of Industrial Innovation, 3rd Edition* (3 ed.). Vol. 1. The MIT Press.

[18] Sean Goggins and Eva Petakovic. 2014. Connecting Theory to Social Technology Platforms: A framework for measuring influence in context. *American Behavioral Scientist* 58, 10 (2014), 1376–1392.

[19] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's data from a firehose. In *9th IEEE Working Conference of Mining Software Repositories, MSR 2012, June 2-3, 2012, Zurich, Switzerland*. 12–21.

[20] Ilya Grigorik. 2012. Github Archive. https://www.githubarchive.org/. (2012). Online: Dec. 5, 2017.

[21] Il-Horn Hann, Jeff Roberts, and Sandra Slaughter. 2004. Why developers participate in open source software projects: an empirical investigation. *International Conference on Information Systems 2004* (2004).

[22] Alexander Hars and Shaosong Ou. 2002. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce* 6, 3 (2002), 25–39.

[23] James Herbsleb, Jason Tsay, Coleen Stuart, and Laura Dabbish. 2013. Leveraging Transparency. *IEEE Software* 30 (2013), 37–43.

[24] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy* 32, 7 (2003), 1159–1177.

[25] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub (Extended Version). In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014)*. ACM, 92–101.

[26] Bruce Kogut and Anca Metiu. 2001. Open-Source Software Development and Distributed Innovation. *Oxford Review of Economic Policy* 17, 2 (2001), 248–264.

[27] Heli Koski. 2007. *Private-collective Software Business Models: Cordinatitons and Commercialization via Licensing*. Discussion Papers 1091. The Research Institute of the Finnish Economy.

[28] Sandeep Krishnamurthy. 2006. On the Intrinsic and Extrinsic Motivation of Free/Libre/Open Source Developers. *Knowledge, Technology and Policy* 18, 4 (2006), 17–39.

[29] Sandeep Krishnamurthy, Shaosong Ou, and Arvind K. Tripathi. 2014. Acceptance of monetary rewards in open source software development. *Research Policy* 43, 4 (2014), 632–644.

[30] Karim Lakhani and Robert Wolf. 2005. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. In *Perspectives on Free and Open Source Software*, Joeseph Feller, Brian Fitzgerald, and Hissam Scott (Eds.). MIT Press, Cambridge.

[31] Michael Lee, Bruce Ferweda, Junghong Choi, Jungpil Hahn, Jae Yun Moon, and Jinwoo Kim. 2013. GitHub developers use rockstars to overcome overflow of news. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. 133–138.

[32] Josh Lerner and Jean Tirole. 2002. Some Simple Economics of Open Source. *Journal of Industrial Economics* 50 (2002), 197–234.

[33] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding Together at Scale: GitHub as a Collaborative Social Network. *Eighth International AAAI Conference on Weblogs and Social Media* (2014).

[34] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. 2013. Impression formation in online peers production: Activity traces and personal profiles in GitHub. In *Proceedings of 2013 Conference on Computer Supported Cooperative Work*. 117–128.

[35] Graziella Marzi. 2009. *If not for money for what? Digging into the OS/FS contributorsâĂŹ motivations*. Working Papers 166. University of Milano-Bicocca, Department of Economics.

[36] Nora McDonald, Kelly Blincoe, Eva Petakovic, and Sean Goggins. 2014. Modelling distributed collaboration on GitHub. *Advances in Complex Systems* 17, 07n08 (2014), 14500–14524.

[37] Regis Meissonier and Isabelle Bourdon. 2012. Toward an Enacted Approach to Understanding OSS Developers Motivations. *International Journal of Technology and Human Interactions* 8, 2 (2012), 38–54.

[38] Ioana Popovici. 2007. *The Determinants of Open Source Quality: An Empirical Investigation*. Working Papers 704. Florida International University.

[39] Ayushi Rastogi and Nachiappan Nagappan. 2016. Forking and the Sustainability of the Developer Community Participation – An Empirical Investigation on Outcomes and Reasons. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 102–111.

[40] Eric Raymond. 1998. Homesteading the Noosphere. *First Monday* 3, 10 (1998).

[41] Eric S. Raymond. 2003. *The Art of UNIX Programming*. Pearson Education.

[42] Georg Von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W. Wallin. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development. *MIS Quarterly* 36, 2 (June 2012), 649–676.

[43] Chorng-Guang Wu, James H. Gerlach, and Clifford E. Young. 2007. An empirical analysis of open source software developers' motivations and continuance intentions. *Information & Management* 44, 3 (2007), 253–262.

[44] Yunwen Ye and Kouichi Kishida. 2003. Toward an Understanding of the Motivation Open Source Software Developers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*. 419–429.

[45] Frances Zlotnick. 2017. GitHub Open Source Survey 2017. http://opensourcesurvey.org/2017/. (June 2017). https://doi.org/10.5281/zenodo.806811