Decentralized Collaboration of Open Source Software Development

Chi Feng

Tepper School of Business

Carnegie Mellon University

Abstract

This dissertation contains three essays on the study of open source software collaboration. The first essay, titled "*Overview of Open Source Software Development*", highlights the link between developer incentives under the non-monetary setting and the sustainability issue in the open source community. Motivated by the well-known fatal flaw *Heartbleed* in the widely used open source network security framework OpenSSL, Chapter 1 discusses the lack of manpower in the open source community from the perspective of developer incentives. I introduce the empirical setting of this project, *GitHub*, one the most popular open source development platforms. Both the past literature as well as the social network features on *GitHub* suggest that the motivations to contribute to open source projects could include (but not limited to) ego stratification, reputation, and career concern.

The second Chapter, "*An Empirical Case Study of Open Source Software Community*", provides descriptive statistics of the contribution history data for Python projects from *GitHub*, and establishes several empirical patterns of open source contribution to showcase the lack of manpower in the community. I find that the distribution of contribution of a project follows power law, in the sense that most projects have very few contributors and number of commits while very few projects have a large number of contributors and commits. In addition, the number of new contributors of a project is positively correlated with its current number of authors and watchers at first, then negatively correlated. For the number of commits, the opposite holds true.

In the last Chapter, titled "*A Structural Model of Decentralized Open Source Software Development*", I build a dynamic discrete choice model characterizing individual developer choice problem in order to understand the sustainability issue in the open source community. Using finite-dependence (Arcidiacono and Miller, 2011, 2019b,a),

I identify and estimate the parameters of an individual developer's utility function. The estimation results show that developers prefer to contribute to popular projects. At the same time, they prefer their own contribution to not be "diluted" by their peers. This is the first study, to the extent of my knowledge, to build and estimate a structural model to build a direct link between developer preferences and the choice of open source contribution. Lastly, given the estimated structural parameters, I conduct counterfactual analysis by increasing the expected popularity of projects except for new projects. The results show that the number of contributors choosing not to commit would decrease under the counterfactual regime.

Acknowledgement

I would like to thank my committee chair Robert Miller, as well as my committee members Bogdan Vasilescu, Rebecca Lessem, Karam Kang for their immense help and support. I also thank my fellow PhD student Mauro Moretto for the inspiring conversations throughout the project. Thanks are also owed to my parents for their support. Lastly, I thank Laila Lee and Lawrence Rapp for their help to facilitate this dissertation. All errors are my own.

Contents

Chapter 1

Overview of Open Source Software Development

1.1 Motivation

Open source softwares (OSS) refer to softwares that can be freely accessed, used, changed, and shared by anyone and for any purpose. Since the 1990s, open source softwares have been widely used by individuals and institutes. Many free and open source projects remain competitive against proprietary softwares. Successful examples include Linux, Apache, MySQL and so on. It is interesting that those softwares can reach commercial quality even though open source projects are usually voluntary-based and not monetized directly.

Despite being used by thousands to even millions of users, the development of a lot of open source projects are not supported by enough manpower. In 2014, many websites and applications were affected by a security vulnerability called *Heartbleed*. This fatal flaw might leak sensitive information including usernames and passwords to third parties. Forbes cybersecurity columnist Joseph Steinberg wrote: *Some might argue that Heartbleed is the worst vulnerability found (at least in terms of its potential impact) since commercial traffic began to flow on the Internet. Heartbleed* was due to a bug in *OpenSSL*, a popular open source security framework that is used by over 66% of internet servers. Being as vital as OpenSSL, it is shocking that there were only one

full-time employee and $2000 per year of donations[1].

Figure 1.1 shows the contribution history of OpenSSL, whose source code is hosted on `github.com`, the largest open source development platform on the internet. On the top left, GitHub plots the total number of commits(a single change to a file or set of files) to OpenSSL in each month from Dec 1998 to Oct 2019. Before 2014 when the vulnerability was discovered, there were less than 50 commits per month to OpenSSL. In addition, among the top 20 contributors[2] of the project, only 5 has made commits to OpenSSL before 2014.

This project intends to answer the following questions. First, how common is the issue of lack of labor force in the open source software community? Second, to understand the labor force, we need to understand how the individual developers choose open source projects to contribute to. Lastly, given the individual contribution incentives, how can we improve the sustainability of the open source ecosystem?

One argument for the success of open source softwares is that any one can contribute freely. So the development of open source softwares would be supported by developers from all over the world. The increasingly large pool of over 40 million developers come from 41 different countries and regions as of 2019[3]. Figure 1.2 shows the trend for the geographic locations of developers on GitHub from 2014 to 2019. There are more and more developers coming from multiple continents working on open software projects.

However, due to the large number of open source projects (currently over 44 million on GitHub), there usually aren't enough contributors for most projects. In fact, more than 90% of Python projects on GitHub has less than 4 contributors and less than 80 total commits (a commit is an individual change to a file or set of files).

To understand the issue of lack of labor support in the open source community, we need to first understand the incentives of open source contributors and how they choose open source projects. By definition, the source code of open source softwares are released under copyrights that allow users to change and distribute the software to anyone. As a result, most of the open source softwares are free of charge and

[1] http://veridicalsystems.com/blog/of-money-responsibility-and-pride/
[2] Developers are ranked by the number of total commits to OpenSSL.
[3] Source: Octoverse, `https://octoverse.github.com`

Figure 1.1: Contribution History By Individual Contributors of OpenSSL

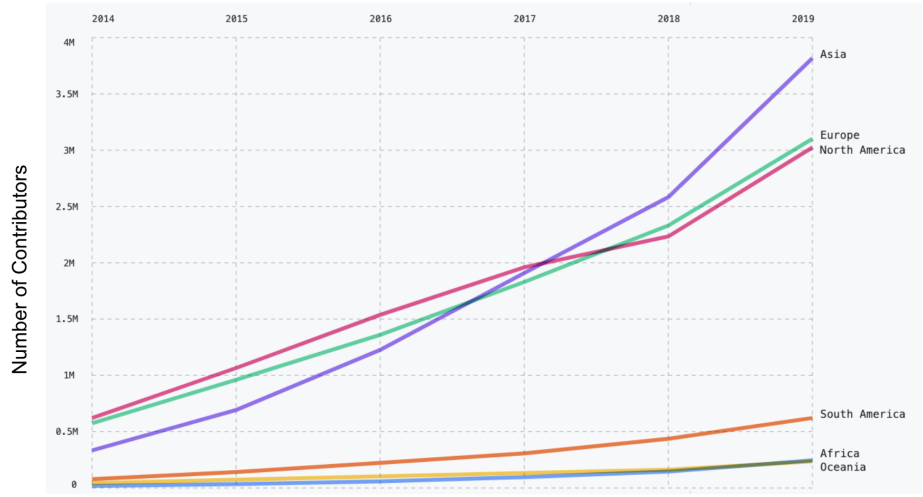

Figure 1.2: Number of GitHub Users (in million) by Geographic Locations

many developers are not compensated monetarily for their work in the open source community.

Based on a survey by GitHub in 2017 (Geiger, 2017), only about 22%[4] of developers on GitHub has some or all of their work duties include contributing to open source projects. Since many open source developers need to work a full-time job that's not directly related to open source projects, they might find it struggling to give timely support and fulfill user demands in the open source community.

Open source community, like all shared-resource system, might suffer from the tragedy of the commons. Because all individuals and institutions can take advantage of open source softwares freely, there is incentive to under-invest in those projects. The under-investment issue is talked about in the landmark report *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure* (Eghbal, 2016) published by Ford Foundation. The author wrote:

> *...many projects are trapped somewhere in the middle: large enough to require significant maintenance, but not quite so large that corporations are clamoring to offer support. These are the stories that go unnoticed and untold. From both sides, these maintainers are told they are the problem: Small project maintainers think mid-sized maintainers should just learn to cope, and large project maintainers think if the project were "good enough,"*

[4]There might be self-selection bias in the survey response. That is, people who respond to the survey might be more active in open source community than other developers.

institutional support would have already come to them.

In addition to the under-investment issue of labor, it is also important to consider if the current distribution of the limited time and effort across different open source projects is efficient. For example, when developers choose a project to contribute to, she would choose based on her personal utility including the signaling value (for example, out of career concern) and cost for joining an existing project. The incentives would not necessarily align with optimization of the open source software production. For example, if a developer find contributing to a sizeable project cost too much effort while has little signaling value, she might prefer to start a new project or join a small project. As a result, certain important projects might lack enough contribution from the open source community.

Therefore, it is important to discuss the incentives of open source developers. Past literature has studied motivations of contributing to open source softwares, including self-use, career concern, reputation, ego gratification, reciprocity, altruism and so on (Lerner and Tirole, 2003; Lakhani and Wolf, 2003; Shah, 2004, 2006; Roberts et al., 2006; von Krogh et al., 2012). In this study, I will draw references from the past literature, and argue that developers care about the popularity of the open source projects since more popular projects have more users and developers, thus can satisfy the developers' ego gratification, reputation and career portfolio building. In addition, they would care about their relative shares of contribution. In each project's page on GitHub, contributors are ranked by their total number of commits. So authors who made more commits are shown higher up and have higher visibility. This is also consistent with reputation, career motivations and ego gratification.

This study will focus on how developers choose projects to contribute to might affect the distribution of labor resources across open source projects. In the first chapter, I will provide an overview of the current state of the open source community and provide the empirical setting for the next two chapters. In Chapter 2, I will present the technical institution background and empirical patterns of the open source community using data from GitHub. In Chapter 3, I will build a dynamic discrete choice model to characterize how individual developers choose open source projects. In addition, I will discuss the identification and estimation of the model. Lastly, using

estimation results, I will conduct counterfactual analysis on the model and discuss the sustainability of the open source community.

1.2 Literature Review

The phenomenon of open source software community has attracted much attention from economists and sociologists for almost 30 years. The discussion mainly focuses on two major questions: why programmers contribute to open source projects when they receive no monetary return but incur various costs including the opportunity cost to compose codes; and how a large group of open source contributers can coordinate with each other. In the seminal paper by Lerner and Tirole (2003), the authors suggest that benefits of learning-by-doing, job market signaling and ego gratification could motivate the programmers to contribute besides altruism. Also, they stated that the fact that software development projects can be divided into smaller and well-defined tasks (modularity) as well as individual leadership, that is, the owner can give certification and recommendations could lead to successful coordination.

Motivations of Open Source Contribution

Since Lerner and Tirole (2003), there emerges many papers that provide both the theoretical and empirical evidences on the issues of developer motivation. Hargrave and Van De Ven (2006) propose a private-collective model of innovation which combines the private investment model through intellectual rights (Demsetz, 1967) and collective-action model where free-riding could happen. It implies that open innovation through private investments can happen when private rewards for the participants are higher than the free-riders. Gächter et al. (2010) showcases a knowledge-sharing game with multiple equilibria, and provides laboratory evidence that sharing knowledge to the public can be fragile compared to concealment.

Empirically, Lakhani and Wolf (2003) analyze data from online survey of 684 software developers in 287 open source software projects. They find that how creative a person feels when they do the project along with user need, intellectual stimulation derived from writing code, and improving programming skills are the main motivations for open source developers. Reputation and career concerns enhances developer

motivation as well.

In a longitudinal study of Apache contributors using survey (Roberts et al., 2006), the authors find that developers' intrinsic motivation to contribute does not diminish in the presence of monetary payment for contributing to the project. In addition, the developers' salary is not affected by the amount of contribution but affected by the rank within the organization. More empirical tests show that open source developers have motivations from both intrinsic and extrinsic sources, such as ideology, fun, reputation, learning, career[5].

In addition, Shah (2006) conducted 88 interviews of open source contributors and collected online documentation data to study the motivations of open source contribution. She finds that developers contribute to open source projects due to reciprocity, desire to integrate one's own code into source code, career concerns, fun and enjoyment.

More recently, Xu et al. (2019) studies the relationship between voluntary contribution in online platforms. They use data from *Stackoverflow*, a large online question-and-answer community. Since a large portion of questions in the platform is programming related, there might be some overlapping of the users of *Stackoverflow* and *GitHub*. Using a difference-in-difference strategy, they show that users contribute 23.7% less in answer and edit activities after they find a new job. Whereas, other activities (including upvoting, downvoting) reduce by 7.4%. This shows that reputation and career concern are important factors that influences voluntary contributions.

Collaboration and Sustainability of Open Source Community

Literature in the strand of the motivations of open source developers explains why a large number of programmers participate in the open source software developement. However, they do not explain what kind of projects or tasks the open source developers choose to contribute and how much effort they devote. Individual developers might tend to choose projects that can signal their skills more, and not necessarily the projects that need the effort most. Thus the effort distribution might not be socially optimal. In addition, given a project, developers might not want to put effort on

[5]For a more extensive review, please see von Krogh et al. (2012)

necessary but "boring" tasks such as compatibility maintenance and debugging. For other tasks such as adding new feature and enhancement, there could be too much effort. Thus, compared to a centralized innovation institution (*e.g.* R&D department in a firm), open source project might face coordination problem in task and effort distribution. In this aspect, this paper relates to the literature on firm control and the necessity of contracts. This can be traced back to Coase (1937) where he proposed that firms exist in the market instead of bilateral co-ordination due to transaction costs. Assuming firms can exert control over workers on their tasks by employment contract, firms can obtain more efficiency than decentralized projects. Therefore, it is important to study the evolution and sustainability of open source community.

Johnson (2002) models open source contribution as a public goods problem. The author models the return for developers as a function of software usage value and development cost, and derives the welfares loss against social planner due to free-riding. Athey and Ellison (2014) models the dynamic development of open source softwares in a public goods framework under altruism. The authors found that quality of open source softwares and the number of contributors follows non-monotone evolution dynamics.

Outside of economic research, there are many studies on the sustainability of open source development from computer science and information system. von Krogh et al. (2003) examined how developers choose to join the open source project and how they specialize by analyzing data from Freenet development process. They find that there's substantial cost associated with contributing to open source projects. Developers, especially newcomers, display specialization behavior (developers maintain the same module instead of multiple modules). Langlois and Garzarelli (2008) argues that open source projects coordinates the division of labor through the institutions of modularity (one can decompose the entire design of a software into relatively independently smaller modules). The trade-off between modularity and its opposite, integrality, decides the threshold for decentralized coordination. As the authors argue, modularity is a necessary condition for open source collaboration, but it doesn't guarantee how well the authors can coordinate with each other, thus how successful and well-maintained the project can be.

Coelho and Valente (2017) uses data of the top-5000 most popular projects on GitHub and conducts survey with developers of 104 popular but deprecated open source projects. Comparing data of non-deprecated projects with deprecated one, deprecated projects are older, smaller in project size (number of commits) and number of contributors. From the survey with deprecated project developers, they find that open source projects could fail due to reasons related to project characteristics (41 projects, *e.g.* low maintainability, outdated technology), or team-related reasons (39 projects, *e.g.* lack of interest), or environment reasons (30 projects, *e.g.* usurped by a competitor or legal issues).

In a related study, Khondhu et al. (2013) compares the characteristics of "active", "dormant" and "inactive" porjects on SourceForge. The authors find that over 93% projects have no recorded activity in the last one year till the observation date (Nov 2012), among which about 14% have been moved to other places. In addition, active projects generally start larger than dormant and inactive ones, and they grow consistently larger.

In addition, Foucault et al. (2015) investigate how personnel turnover can influence the quality of open source projects (in terms of bugfixes) using 5 popular open source projects. They find that newcomers of a project have a negative effect on the quality of a team's work, while leavers do not have such effect.

More recently, Valiev et al. (2018) study the relation between a project's position in the dependency network (for example, whether the other packages depend on it or the package depends on other packages) and the likelihood of a project going into dormancy using Python packages and their GitHub contributor information. The authors find that projects depending on more packages has higher chance to be dormant in early stages of development, but lower chance in later stages. In addition, the number of contributors and core-developers positively correlates with the survival rate of a project.

In addition, Qiu et al. (2019) combines contributor survey and GitHub activity data to study how signals of project attractiveness can influence the decision of joining in a GitHub project. Important factors that positively affect the incoming of new contributors include how actively maintained and popular the project currently is, as

well how friendly and responsive the current maintainers are. Also, Miller et al. (2019) investigate the disengagement of open source contributors using developer survey data and their GitHub contribution data. They find that contributors who work on more popular projects are less likely to disengage, controlling for their working hours and total contribution activities.

Lastly, Fronchetti et al. (2019) use the contribution data of 450 open source projects on GitHub, and find that the popularity of the project (in terms of stars), time to review pull requests, project age, and programming languages are the factors that best explain the newcomers' growth patterns. Also, Borges et al. (2016) collects historical data of 2500 popular projects on GitHub. Unconditional on the ownership of a project, there is no correlation between numbers of stars and the project's age since organization-owned projects usually gain popularity faster than individually owned projects. Also, there is a weak correlation between popularity and the number of commits and contributors. In terms of popularity growth, repositories tend to receive more stars right after their first public release. Afterwards, the growth rate stabilizes for half of the repositories.

1.3 Empirical Setting and Data

Since the movement of open source softwares in the 90s, many platforms emerged to smooth the coordination by storing project files, facilitate communications among team members. A famous example is *SourceForge.net*, an open source software hosting platform launched in 1999. On SourceForge, owners have absolute control of developing and uploading projects. If other developers would like to contribute, they need to download the entire set of code files and make modifications. However, the emergence of *git* technology makes decentralized collaborations among developers much easier and straightforward with "forking"[6], "pull requests"[7] and "merging branches"[8]. Git sets the foundation of the currently most popular way of open source development. It allows scalable decentralized coordinating work among programmers by creating multiple branches of the main projects and tracking changes in files. In this study, I use the contribution activity data from GitHub, one of the largest open source development platform that utilizes *git* technology. In the following section, I will set the technical background and introduce the data for this study.

The dataset collection is provided by *GHTorrent* (Georgios Gousios, 2013) who collects commit activities of all public projects from GitHub API (current observation date is July 2017)[9]. In the data, there are 30.08 million original projects (excluding forked projects, 27.38m owned by individuals and 2.70m owned by organizations) and 16.49 million users (15.84 million individual users and 0.65 million organizations).

1.3.1 Git-based Software Development

Figure 1.3 summarizes the workflow on GitHub[10]. The collaboration on GitHub is completely decentralized, in the sense that one can make a change to a project without affecting the original project (master branch). One can do so by forking a project,

[6]A fork is a copy of a project. Forking a projects allows one to freely experiment with code changes without affecting the original project.

[7]A pull request occurs when a developer asks for changes committed to a forked project to be considered for inclusion in the main project.

[8]Merge command lets one take the independent strands of development created by git branch and integrate them into a single branch.

[9]For a more detailed description of the interrelation data, see `http://ghtorrent.org/relational.html`

[10]`https://arccwiki.uwyo.edu/index.php/Git\_Workflow\#GitFlow\_and\_GitHub\_Flow\_differences`

that is, copying a project to one's own profile and make changes on it (commits). If a developer would like to propose a change to the original project, he/she can fork the project, make changes and then make a pull request to the original project. When this branch is ready to be merged back into master, she can open a pull request which will be reviewed and tested by the project's collaborators. If the pull request is approved, it will be merged into master. This kind of workflow design minimizes the code conflicts during collaborative development.

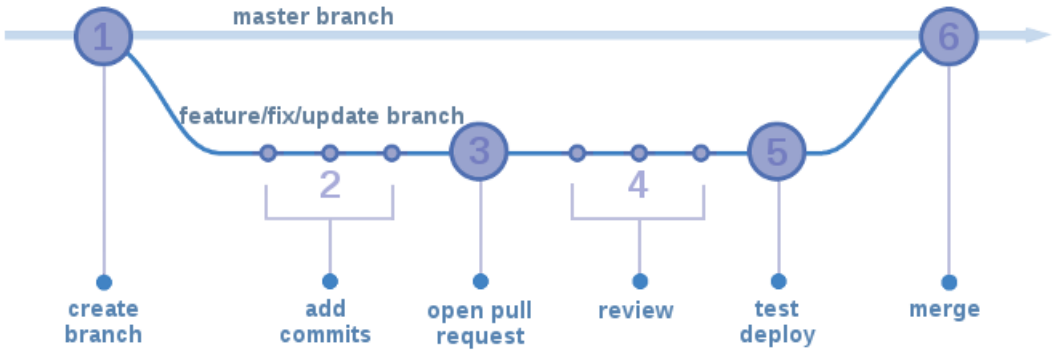

Figure 1.3: GitHub WorkFlow

The unit for code contribution on GitHub is called "commit". It is an individual change to a file or a set of files. We can observe the identity of the user who makes the commit, which project he/she make commit to and when he/she makes it. For each project, users can raise "issues" to request a bug fix, more features or propose enhancements to a project. An user is not required to provide code when raising an issue. However, if one wish to contribute code modifications to the project, he/she can make a pull request. All users can comment on issues and pull requests.

1.3.2 Social Nature of GitHub

Another distinct feature of GitHub that are different from other open source software platforms such as *SourceForge.net* is its social network nature. On GitHub, each individual has a profile page (for instance, Figure 1.4 [11]) which summarizes one's projects, recent commit history as well as some individual information such as organization, contact information. Note that many developers use their real names on GitHub.

[11] Screenshot in Nov 2018

Many even put their GitHub profile on their resumes[12].

In addition, if one clicks on the "Contributor" tab of a project's homepage, we can find the authors of the project listed by the number of commits they made. Figure 1.5 is an example for Numpy, one of the most widely used library in Python[13].

Another "social network" feature of GitHub is ability to "star" a project (originally "watch"). Similar to liking posts on *twitter*, starring a project shows appreciation of the project. There are multiple studies investigating the implications of "stars" on GitHub (*e.g.* Borges and Tulio Valente (2018), Fronchetti et al. (2019)). The results show that the number of stars is a metric for popularity, and can influence the future contribution of a project.

These "social network" features including individual developer's personal profiles as well as "stars/watchers" enhances the motivations of ego gratification, reputation and career concern to participate in the open source community. Anecdotally, many developers use their GitHub profile as their "second resume" when they look for jobs because their public projects can showcase their skills[14].

[12]Anecdotally, see for example `https://medium.com/@sitapati/the-impact-github-is-having-on-your-software-career-right-now-6ce536ec0b50` and `https://workplace.stackexchange.com/questions/24128/should-i-include-my-github-page-on-my-resume` where software engineers and recruiters argue why it might be a good idea to include one's GitHub profile in the resume

[13]Screenshot in Nov 2018

[14]For example, this blog (`https://thehftguy.com/2016/10/24/heres-how-to-make-a-good-github-project-for-your-resume/`) gives a tutorial to present GitHub projects to recruiters.

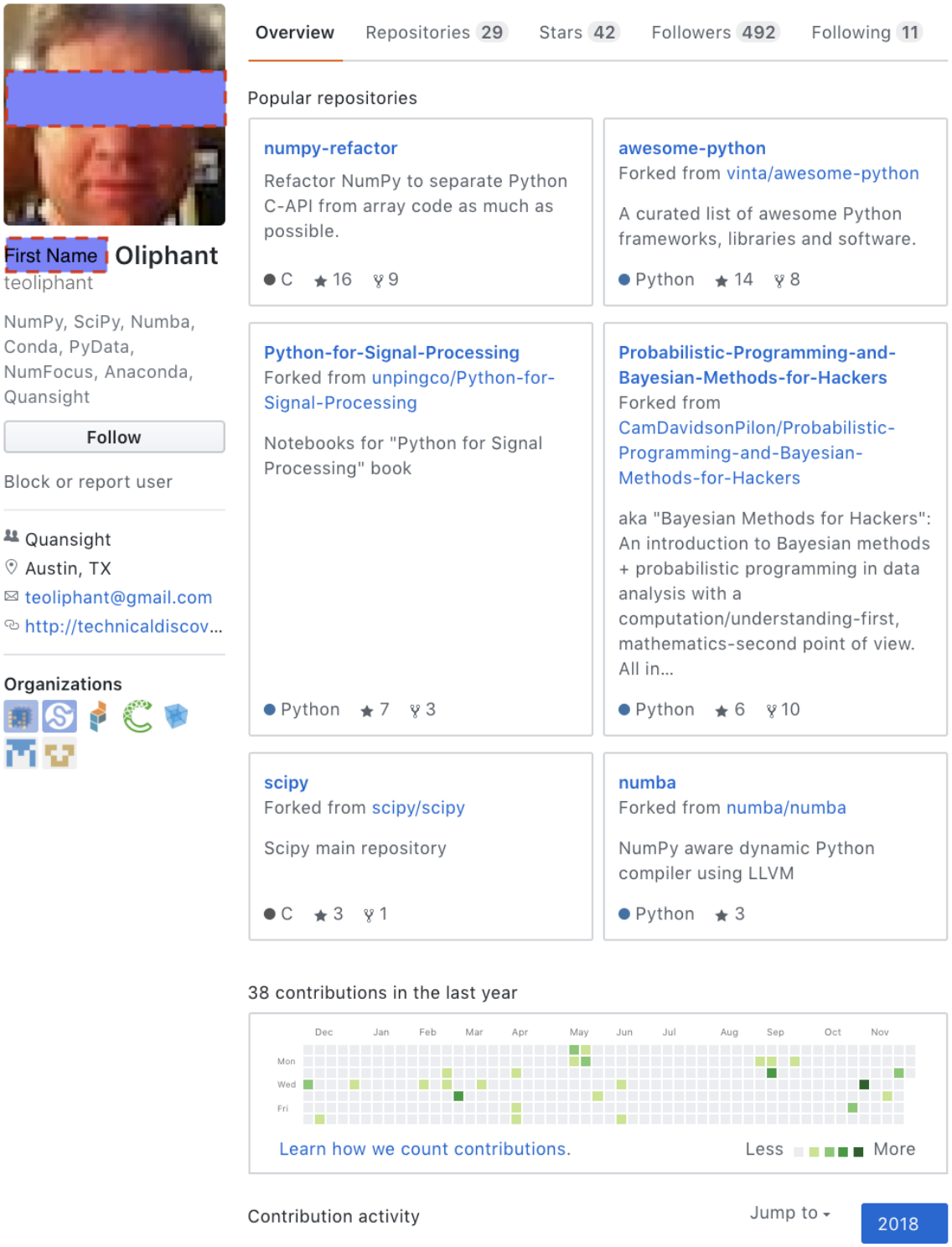

Figure 1.4: Sample GitHub Profile

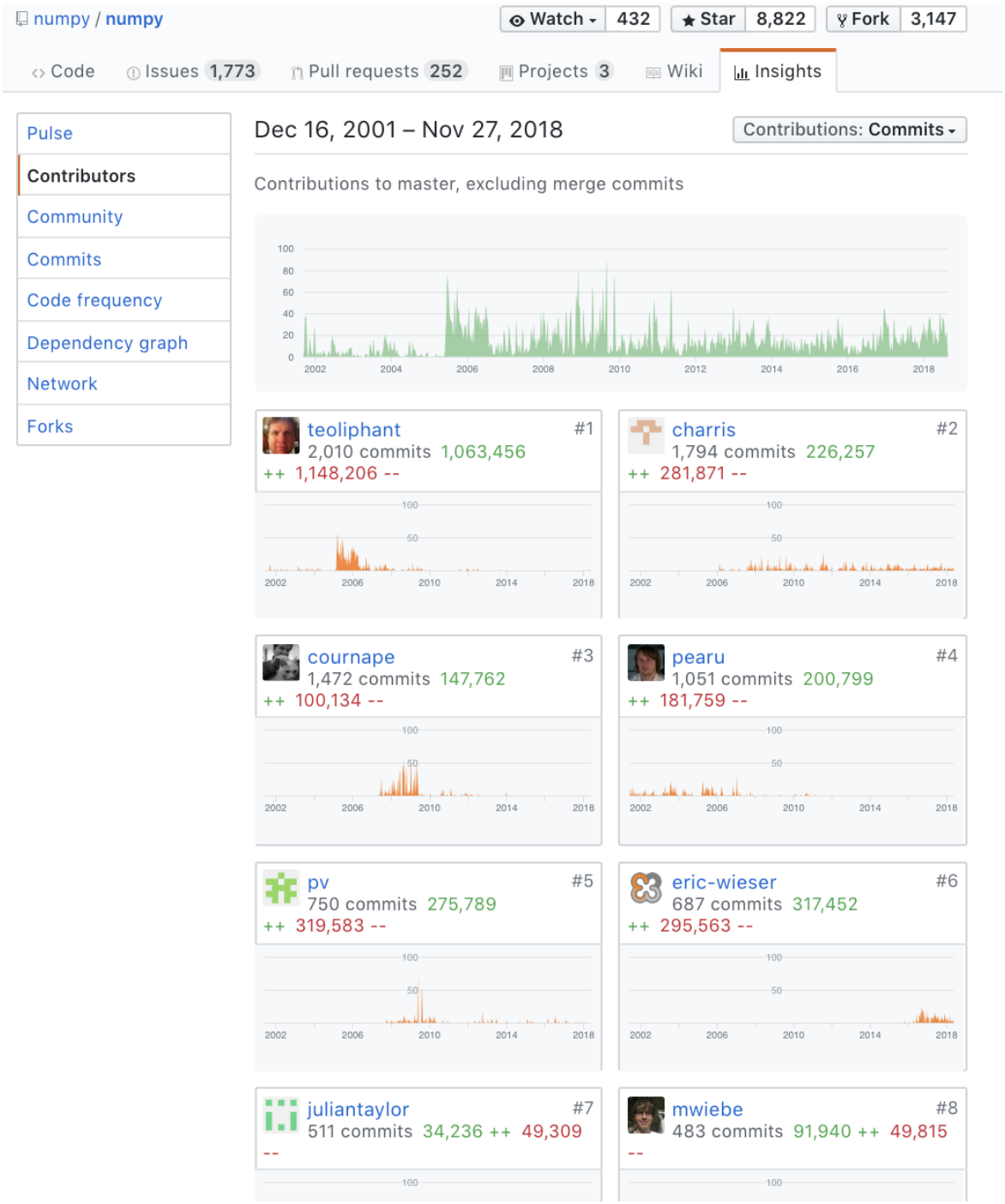

Figure 1.5: Top Authors of Project Numpy

1.4 Conclusion

Free and open source softwares have changed how people develop and use new technology since the 1990s. The internationally recognized definition by Open Source Initiative states in the first criteria of open source that

> *Open source doesn't just mean access to the source code. The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.*

That is, open source software are free to distribute by nature. The non-monetary setting is different from the traditional business model of propriety softwares, and different from the usual object of discussion within the field of economics.

Despite the emergence of many successful open source projects including MySQL and Linux who are even strong enough to challenge their mainstream propriety competitors, the issue of sustainability of open source softwares has raised concerns from its community. In 2014, the fatal flaw *Heartbleed* found in OpenSSL, a widely used open source network security framework, has exposed the lack of manpower in open source projects.

Due to the non-monetary nature of the open source community, we need to further understand the motivation and mechanism of open source contribution in order to discuss the issue of sustainability of open source softwares. Many past studies have explored the motivations for developers to contribute to open source projects and the patterns of collaboration in the community. However, there is not much discussion on how personal incentives of developers influence the development of open source softwares thus the distribution of open source projects. This study recognizes the direct link between individual developer's motivation in developing open source softwares and the sustainability of open source community.

Past studies argued that individual developers contribute to open source softwares out of the following reasons: altruism and reciprocity, ego gratification, reputation as well as career concerns. Here I focus on GitHub, one of the most popular open source

software development platforms. GitHub tracks the contribution history of each individual and each project. In addition, it has several "social network" features including "stars/watchers" which enhances the motivations of ego gratification, reputation and career concern. This study will build on past literature and empirical patterns from GitHub to examine the motivation of developers, and build a link between developer incentive and sustainability of open source community.

Chapter 2

An Empirical Case Study of Open Source Software Community

2.1 Introduction

The second chapter gives the technical background of GitHub and provides descriptive analysis of open source software contributions using data from GitHub, one of the most popular open source project development platforms. Here I try to answer the following questions: Is it true that most of the open source softwares lack manpower and thus not well-maintained? In addition, how do the developers choose the projects to contribute to?

Specifically I will show that the distribution of contribution is highly skewed across different projects. On top of that, unsurprisingly, the time and effort of most developers are very limited since many developers do not get paid directly from open source contribution and open source softwares are a type of public goods.

2.2 Institution Background

In this study, I use the individual contribution data from GitHub whose workflow makes it suitable for decentralized collaboration. When an agent starts a project, she will open a repository. The repository can be public or private. Private repositories are hidden from public and can be only seen by their owners and authorized contributors,

thus are not part of the current data. This data contains only public repositories which can be seen by anyone.

Additionally, anyone can fork public projects, make commits and make pull requests. If the pull requests are accepted by the owner of the project, then the changes will be merged into the main branch of the project. Below I summarize the main terminologies that I use in this study:

- **Project and Repository**: A repository is a folder containing all project files, and stores their revision history. Repositories can have multiple collaborators and can be either public or private.

- **Commit**: An individual change to a file (or set of files). Each commit generates a unique id with a timestamp.

- **Watch/star**[1]: A feature on GitHub for users to show appreciation of a project. Here I use the number of watchers/stars as a measure of popularity for the project.

In this study, I use the contribution records from Python community. Python is one of the most popular programming language in recent years. Figure 2.1 shows the relative ranking of languages used on GitHub from 2014 to 2019 in terms of unique contributors to public and private repositories[2]. Python ranks from the fourth most used language in 2014 to the second in 2019. In addition, compared to other programming languages such as C and Java which do not have a centrazlied package release repository, Python has an official package distribution repository *Python Package Index* (also known as *Pypi*), which makes it easy to track the release of Python packages.

[1]Here I use watch and star interchangeably per *GHTorrent*.
[2]Source: Octoverse, `https://octoverse.github.com`

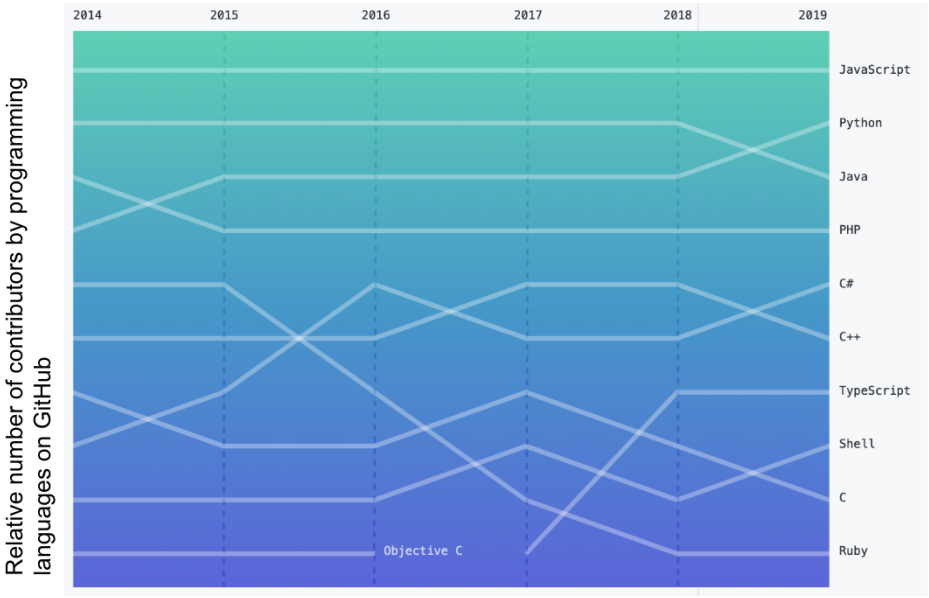

Figure 2.1: Trend of Top Programming Languages on GitHub

2.3 Data Cleaning and Data Description

In this study, I focus on the Python community on GitHub[3] between Oct 2010 and June 2018. Notice that GitHub is a site for development not software distribution. Python packages' distribution are easier to track because most Python packages use PyPI[4] as the distribution site, in contrast to other languages where the actual distributions are based on the software's own separate page.

Here the Python projects in the sample are collected using all projects which has "Python" in the project description (programming language not restricted to Python) as well keywords including "library" and "module" so that I can exclude the personal projects (such as personal websites) or class projects that are not intended to be released as a package. Under this definition, there are 30549 projects and 45353 authors. Among those authors, over 91% of them (41529 authors) contribute to only one project per month. Those authors are responsible for 24345 projects (over 79%) and 734469 unique commits (over 58%). In this project, I will focus on those 91% contributors who contribute to at most one project in a month. The dataset is linked by four main identifiers: project id, author id, commit id and watcher id. Commits and watch actions are also linked with timestamps at which the user makes the commit or

[3]Complete dataset collected by GHTorrent (Georgios Gousios, 2013).
[4]Python Package Index

press the "watch" button. For authors, one can also observe when the author registers to GitHub. Table 2.1 gives a summary of observed variables in the data.

Table 2.1: Observed Variables

commit id	a change to the project
commit time	time of contribution
project id	project that the commit is added to
author id	the developer who made the commit
register time	when the author registers to GitHub
watcher id	follower of the project
watch time	time of watching/starring the project
download count	number of downloads at certain time

Figure 2.2 depicts the trend of the total number of projects in the sample as well as the number of new projects that was just created by month. The blue curve shows the total number of available projects divided by 10 (so that it's on the similar scale as the number of new projects), while the orange curves shows the number of new projects. The number of new projects has a slight increasing trend, thus the total number of available projects has an exponential increasing trend.

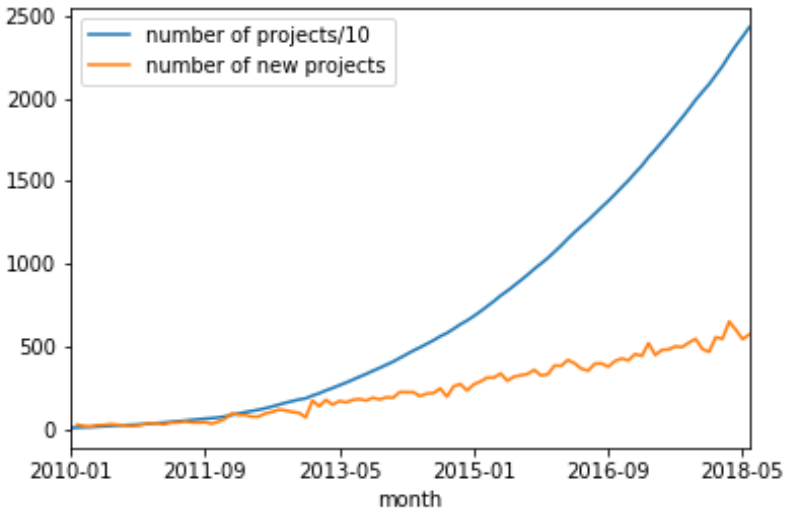

Figure 2.2: Number of Projects By Month

2.4 Empirical Patterns

In this section, I will provide several empirical patterns that will confirm the motivations of this study and guide the model specifications in the next chapter. Figure 2.3 shows the trend of the status of available projects by month. Overall, it shows that the median values of a project's total number of past commits/authors/watchers are below the average from 2010 to 2018. It means that the distributions of project status are highly skewed with the majority of projects having few number of authors, commits and watchers.

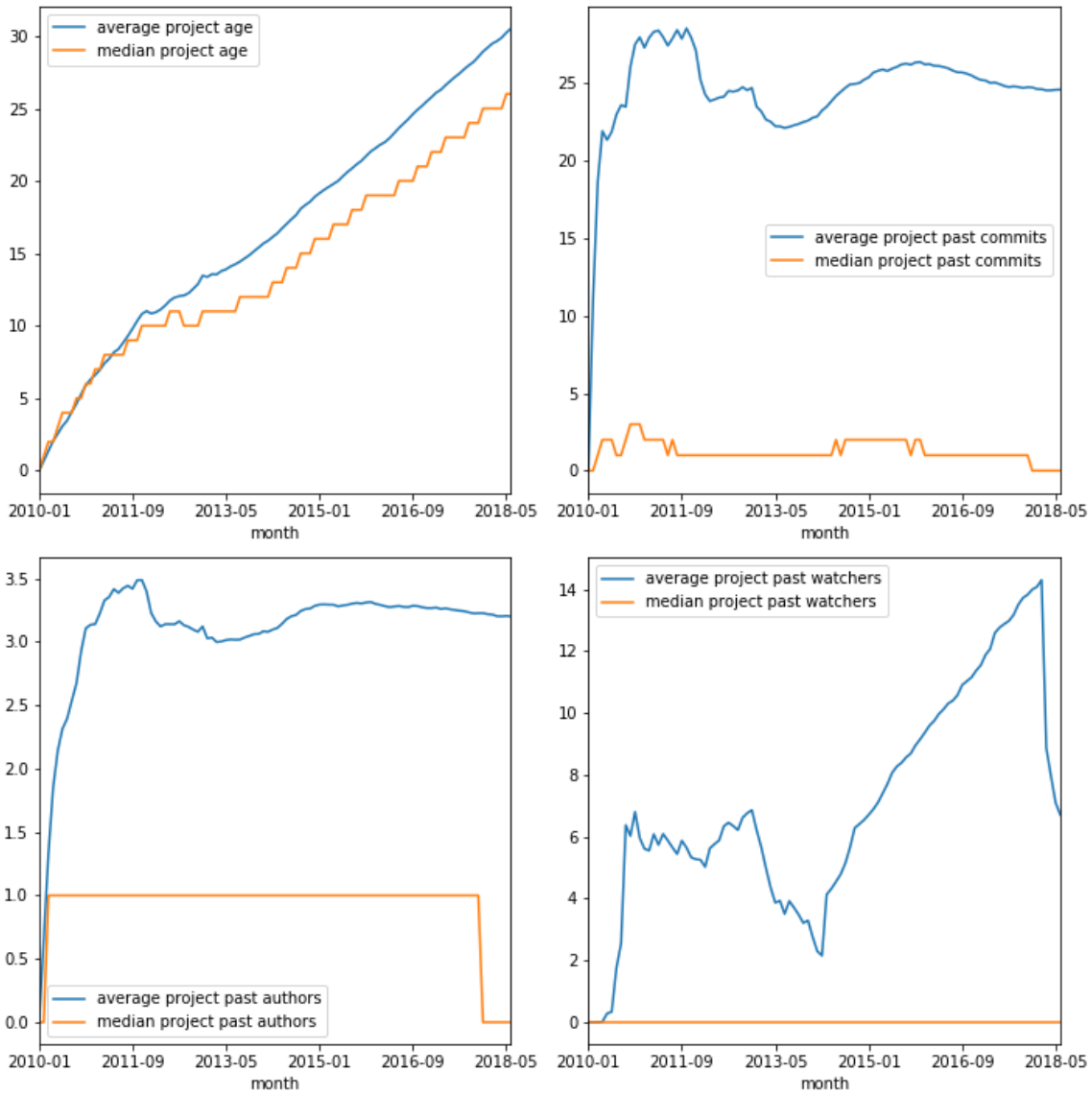

Figure 2.3: Average Project Characteristics By Month

2.4.1 Project Characteristics

In this section, I will analyze the empirical patterns on project characteristics, including the number of authors, the number of commits, and the number of watchers of the Python projects in the data. Table 2.2 provides the descriptive statistics for Python projects based on the commit histories up till June 2018. We can observe that over 90% of projects have less than 10 authors while the maximum number of authors in a project is as high as 1108. The distribution of the number of commits and watchers are also highly skewed.

Table 2.2: Project Descriptive Statistics

	project age	total authors	total commits	total watchers
count	24345	24345	24345	24345
mean	30.53	4.33	30.17	15.02
std	23.08	18	156.03	232.63
min	0	1	1	0
10%	4	1	2	0
50%	26	2	7	0
90%	64	8	54	5
max	101	1108	12842	23304

Note: The observations are based on the most recent record to the data observation date, June 2018.

Since the data consists of projects of different ages, it might not be fair to compare projects that have been created on the platform many years ago versus projects that are created recently. Below in Table 2.3 and 2.4, I decompose the distribution of project contribution by projects' ages in years. Even after decomposing by project age, we can see that the number of authors and the number of commits in a project have very skewed distribution. Most projects receive very little contribution while very few projects receive massive amount of contribution.

The dataset can be re-constructed into panel structures for projects based on the timestamps of commits. Table 2.5 provides summary statistics for projects' panel data by projects' age (months since the creation of the project). Using the panel data of projects, I run a fixed effect regression of a project's new authors within a certain

Table 2.3: Summary Statistics of Number of Authors by Project Age

project age (year)	count	mean	std	min	25%	50%	75%	max
0	112063	2.57	3.41	1	1	2	3	306
1	95022	4.52	7.89	1	1	2	5	450
2	76096	6.14	15.58	1	1	3	6	800
3	54833	7.60	18.47	1	1	3	7	783
4	36266	9.39	22.81	1	2	3	9	704
5	20624	11.92	35.64	1	2	4	11	1057
6	10215	15.64	53.78	1	2	5	13	1108
7	4415	24.70	88.64	1	2	7	19	1104
8	717	34.14	100.57	1	3	12	32	1072

Note: The summary statistics are conditional on project having at least 1 commit. Project ages are rounded up to the closest numbers (in year) under.

Table 2.4: Summary Statistics of Number of Commits by Project Age

project_age_year	count	mean	std	min	10%	50%	90%	max
0	112063	23.53	49.83	1	2	9	55	1672
1	95022	37.91	94.76	1	2	13	87	3652
2	76096	48.23	150.27	1	2	14	103	6266
3	54833	57.22	187.32	1	2	15	121	6928
4	36266	68.54	248.68	1	1	16	138	7420
5	20624	80.81	354.73	1	1	17	154	9761
6	10215	102.08	488.22	1	1	19	185	11071
7	4415	159.45	737.57	1	1	27	291.60	12820
8	717	265.31	1197.39	1	2	53	523.40	12842

Note: The summary statistics are conditional on project having at least 1 commit. Project ages are rounded up to the closest numbers under.

month conditional on the project's current characteristics. Specifically, I have:

$$\Delta\text{authors}_{jt} = \beta_0 + \beta_1 \text{project age}_{jt} + \beta_2 \text{project past commits}_{jt} \tag{2.1}$$

$$+ \beta_3 \text{project past authors}_{jt} + \beta_4 \text{project past watchers}_{jt} + a_j + \eta_{jt} \tag{2.2}$$

where $\Delta\text{authors}_{jt}$ is the number of new authors that project j gets at month t. $\text{project past commits}_{jt}$ is the number of total commits project j gets up till t. Similarly, $\text{project past authors}_{jt}$ and $\text{project past watchers}_{jt}$ are the number of total authors and watchers that project j gets up till month t. a_j is the unobserved project fixed effect, such as the intrinsic quality of project which could be observed by agents by not the

econometrician. Lastly, η_{jt} is the exogenous shock which is *i.i.d.* across project and time.

Table 2.5: Summary Statistics of Project Panel Data

	count	mean	std	min	10%	50%	90%	max
project age	734678	24.15	19.51	0	3	20	52	101
current commits	734678	1	7.21	0	0	0	1	808
current authors	734678	0.14	0.69	0	0	0	1	152
current watchers	734678	0.34	11.77	0	0	0	0	5623
past commits	734678	25.05	145.38	0	0	1	50	12841
past authors	734678	3.23	15.02	0	0	1	7	1104
past watchers	734678	9.79	154.04	0	0	0	0	22770
released	734678	0.18	0.38	0	0	0	1	1

Note: Out of 24345 unique projects, the total number of projects that are released up till June 2018 is 3638.

Table 2.6 gives the estimation results for the fixed effect regression in Equation 2.2 using the project panel data. Column (1) gives the results for the basic specification, while Column (2) gives the specifications including the square terms. From the results in Column (2), we can see that there exists non-linear relationship between the number of new authors and the project's current characteristics. The number of new authors of a project at time t decreases with the number of past commits up to 50 commits. For projects with more than 50 commits, the number of authors increases with the number of past commits. On the contrary, the number of new authors first decreases the number of past authors for the first 1250 authors, then decreases. Similarly, the number of new authors first decreases with the number of past watchers for the first 50 watchers, then decreases.

Table 2.6: Project Fixed Effect Regression

	Dependent variable:	
	project current authors	
	(1)	(2)
project age	-0.006^{***}	-0.016^{***}
	(0.00004)	(0.0001)
project age sq		0.0001^{***}
		(0.000001)
project past commits	-0.0002^{***}	-0.001^{***}
	(0.00001)	(0.00002)
project past commits sq		0.00001^{***}
		(0.000001)
project past authors	0.012^{***}	0.025^{***}
	(0.0001)	(0.0002)
project past authors sq		-0.00002^{***}
		(0.000001)
project past watchers	-0.0001^{***}	0.001^{***}
	(0.00001)	(0.00001)
project past watchers sq		-0.00001^{***}
		(0.000001)
Observations	734,678	734,678
R^2	0.048	0.086
Adjusted R^2	0.016	0.055
F Statistic	$9{,}029.776^{***}$	$8{,}387.571^{***}$
	(df = 4; 710329)	(df = 8; 710325)
Note:		*p<0.1; **p<0.05; ***p<0.01

2.4.2 Contributor Activities

In this section, I will take a closer look of the contribution activities to Python projects on GitHub. Table 2.7 gives the summary statistics for contributors based on the commit histories up till June 2018. We can see that about 90% of authors have only contributed to one project in the past. Therefore in the next Chapter where I build a structural model to characterize the choice decisions of individual developers, I limit an author's choice set to include up to 2 past projects based on the number of commits in those projects.

Table 2.7: Contributor Descriptive Statistics

	author age	current commits	past commits	past projects
count	41514	41514	41514	41514
mean	31.23	8.40	13.53	0.47
std	28.60	16.91	69.01	0.60
min	0	1	0	0
10%	0	1	0	0
50%	25	3	0	0
90%	74	20	27	1
max	123	701	5051	8

Note: The observations are based on the most recent record to the data observation date, June 2018.

Similar to projects, we can restructure the data based on timestamps of commits to get a panel data structure for authors' contribution history. Table 2.8 provides summary statistics for authors' panel data by authors' age (months of registration on GitHub).

Figure 2.4 plots the trend for entries of new authors and new projects in the data. New authors are users who just registered to GitHub in each month. New projects are the projects that had their first commit in a certain month. We can see that the number of new projects is growing each month, whereas the number of new authors is not. So there is less and less number of authors per project. On the other hand, Figure 2.5 plots the exit rate for authors and projects. Here I define the exit of author and project to be having no commits for over a year. The exit rate is calculated by dividing the number of exited authors/projects with total number of authors/projects in the same cohort. Therefore, cohorts of authors who joined more recently and projects

Table 2.8: Summary Statistics of Author Panel Data

	author age	current commits	nonzero commits	past projects
count	1308083	1308083	105387	1308083
mean	28.80	0.56	6.96	0.14
std	22.54	4.49	14.33	0.40
min	0	0	1	0
10%	4	0	1	0
50%	24	0	3	0
90%	61	0	16	1
max	123	701	701	8

Note: Total number of unique authors in the data is 41514. "Author age" refers to the number of months since the author registered on GitHub. The column "nonzero commits" refers to the number of commits that author made in a month *conditional* on having made at least 1 commit.

which started recently will have truncation issue. To reduce the truncation issue, I only plotted authors/projects cohorts up until 2016. The exit rate is fairly constant across cohorts. Combining with Figure 2.4, we can see the changes in authors and projects are mainly dominated by entries rather than exits.

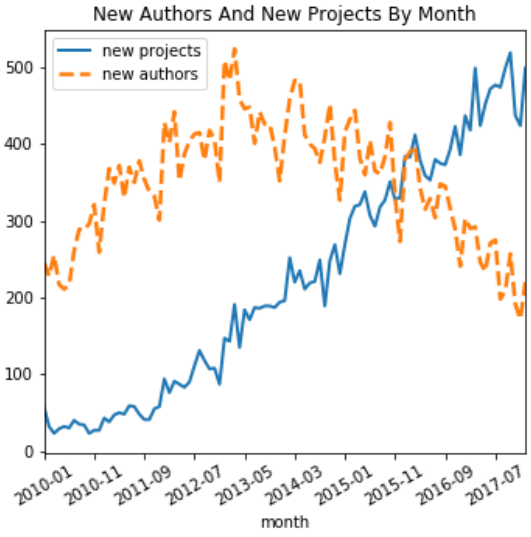

Figure 2.4: Entry of Authors and Projects

Figure 2.5: Exit of Authors and Projects

Figure 2.6 summarizes the contribution intervals by individual developers. Specifically, the left panel plots the month interval between an author's first commit and her last commits by cohort[5]. Since the younger cohorts are more prone to truncation if

[5]When the author first register to the platform.

the cohorts are closer to the data observation time, we will focus on the older cohorts. For example, cohorts from 2008 to 2011 spend 20 months on GitHub on average. The right panel plots the month interval between 2 consecutive commits of an author. For example, author cohorts from 2008 to 2011 contribute every other 4 to 6 months on average. Figure 2.6 shows that individual contributors' time and effort are very limited.

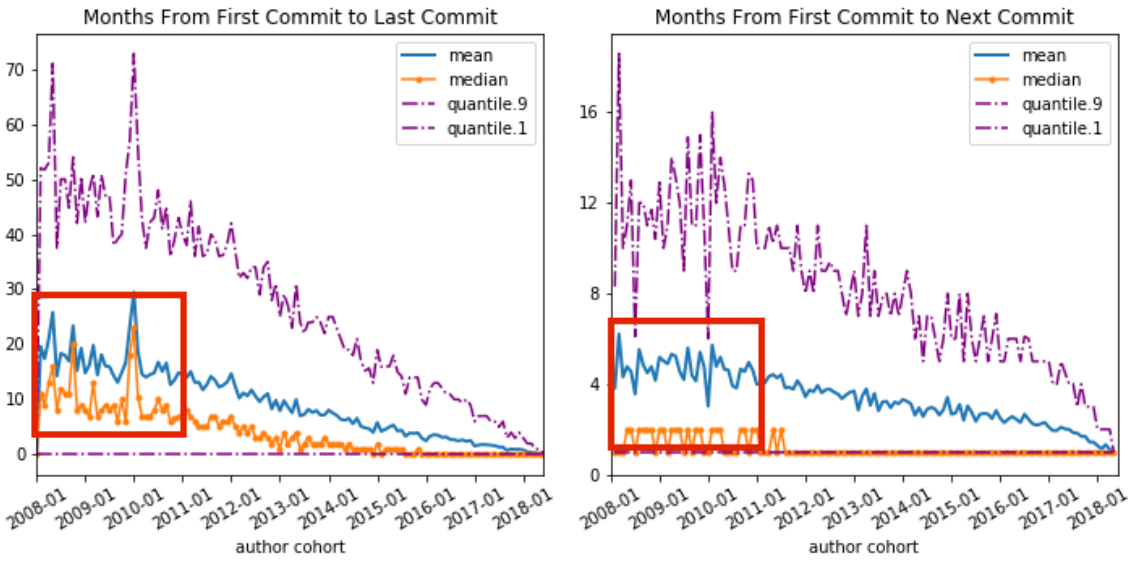

Figure 2.6: Author Contribution Time Intervals

Figure 2.7 further decomposes the project distribution by project age. On average, a project gets more commits and authors as the project age increases. We can still observe the pattern of skewed distribution in the graph. The green line represents the top 10% quantile of the number of commits that are contributed (or the number of active authors who contributed in the right graph) when the project is at a certain age. The blue line, orange line and purple line refer to the mean, the median and lowest 10% quantile of the number of commits (or number of authors in the right graph) at each project age.

Next I further investigate the idea that authors prefer to be a core developer using revealed preference in Table 2.9 and Table 2.10.

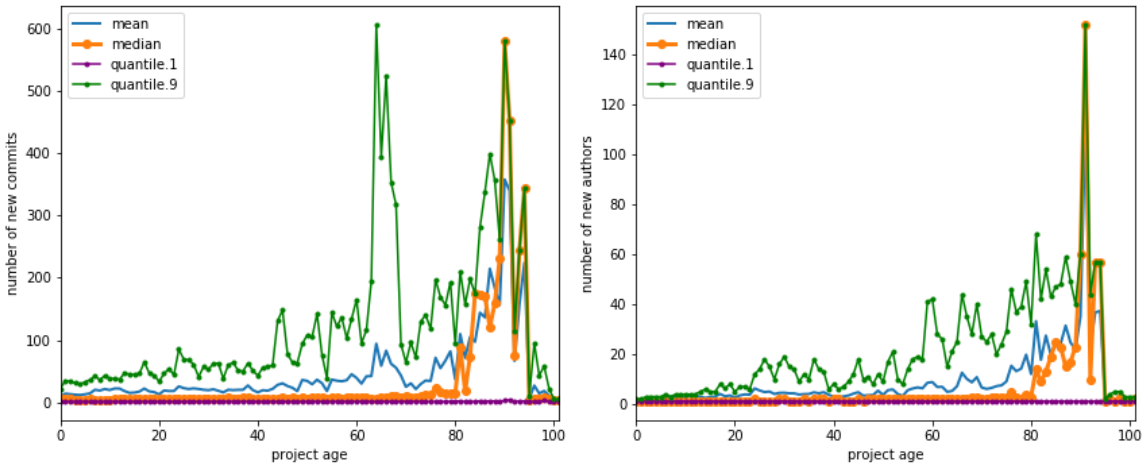

Figure 2.7: New Commits and Active Authors By Project Age

Table 2.9: Comparing Choices for Authors with One Past Project

	Actual Choice		Counterfactual Choice	
	chosen project watchers	% commits	past project watchers	% commits
create new project	0.00	95	263.48	57
join new project	517.34	43	563.00	33
past project	223.94	67	223.94	67

Table 2.10: Comparing Choices for Authors with Two Past Projects

	Actual Choice		Counterfactual Choice		Counterfactual Choice	
	project watchers	% commits	past project 1 watchers	% commits	past project 2 watchers	% commits
create new project	0.00	96	100.77	67	199.36	45
join new project	452.05	43	352.14	41	601.74	31
past project 1	189.06	72	189.06	72	378.72	40
past project 2	235.68	70	224.33	62	235.68	70

2.4.3 Project Popularity, Release and Download

In this section I test if the popularity of a project increases with the number of commits and contributors conditional on project age and project fixed effect. The popularity of a project is measured by the number of watchers of a project at a given point of time. On GitHub, a user can "follow" the development of a project by "watching" or "starring" the project. After that, the user will receive notifications of the project when there are new commits, issues or pull requests. Developers "star" or "watch" a project to show appreciation or bookmark a project (Borges et al., 2016). Thus the number of watchers of popularity can reflect its quality and usefulness. To control for project fixed effect, I use the panel structure of the data.

Table 9 shows the OLS regression results for project popularity. Both the number of commits and the number of contributors (authors) are positively correlated with the

popularity of a project, and the positive correlation is bigger in scale for individually-owned projects than organization owned projects.

Table 2.11: Logit Regression of Project Release

const	-3.6142***	org	1.6641***		
	(0.2169)		(0.0160)		
age	-0.0557***	org age	-0.0082***	age sq	0.0007***
	(0.0011)		(0.0008)		(0.0000)
num commits	0.0051***	org num commits	-0.0014***	num commits sq	-0.0000***
	(0.0000)		(0.0000)		(0.0000)
num authors	0.0463***	org num authors	-0.0015	num authors sq	-0.0006***
	(0.0036)		(0.0031)		(0.0000)
num issues	-0.0040***	org num issues	-0.0022***	num issues sq	0.0000***
	(0.0003)		(0.0004)		(0.0000)
num reporters	0.0114***	org num reporters	0.0027***	num reporters sq	-0.0000***
	(0.0006)		(0.0006)		(0.0000)
year month F.E.	Y	project F.E.	N	no. obs.	6723782

Note: This regression pools all the panel data of release history of all data in the sample, resulting in 6723782 observations covering 307909 projects. There are 11347 packages that had at least one release.

From the release repository of Python packages *Pypi*, there are 116687 packages with at least one release as of January 2018[6]. After deleting project foundries (repositories which holds multiple different foundries), there are 71903 packages that can be matched back to GitHub. Download data for those packages goes from Jan 2016 to Aug 2018.

In this paper, I treat project release as an important milestone for projects. If a project is formally released, it means that the project is relatively complete (not necessarily mature) for users to utilize. Therefore, whether a project is released and how long it takes for a project to be released are important factors to influence social welfare. In the sample of Python projects on GitHub, only 15744 projects are released out of 315745 projects in total (4.99%). The average age of a project when it is released is 3.25 months.

Here I will first conduct a logistic regression of a formal release in Pypi on the project age, the number of commits and authors, and whether the project is owned by an individual or an organization. The data is a cross-section data of all projects at the observation date (Jan 16 2018). Then I will conduct a duration analysis by fitting a Cox Proportional Hazard model to see which factors contribute to faster releases.

[6]I found less projects in my sample, because not all packages host their code on GitHub and not all of them have project descriptions including the words "library" and "module".

Notice here that the number of commits and the number of authors of a project can be affected by the project's innate quality (such as how good the idea is) which is unobserved. So the estimate can be biased.

Table 2.12: Cox Proportional Hazard Regression with Time-varying Variables

	coef	exp(coef)	se(coef)	z	p
num commits	0.0032	1.0032	0.0004	9.1188	0.0000 ***
num authors	0.0633	1.0654	0.0214	2.9632	0.0030 **
num issues	0.0025	1.0025	0.0024	1.0516	0.2930
num reporters	0.0110	1.0110	0.0039	2.8221	0.0048**
num commits sq	-0.0000	1.0000	0.0000	-5.1157	0.0000***
num authors sq	-0.0051	0.9949	0.0013	-3.9570	0.0001***
num issues sq	-0.0000	1.0000	0.0000	-1.0373	0.2996
num reporters sq	-0.0000	1.0000	0.0000	-2.6116	0.0090**
Likelihood ratio test = -17353.520 on 8 df					

To analyze the duration of time before a project is released, I use Cox proportional hazard model. That is, we assume that the hazard for project j to release at t is:

$$\frac{f(t|X_{jt})}{1-F(t|X_{jt})} \equiv \lambda(t|X_{jt}) = \lambda_0(t)\exp(X'_{jt}\beta)$$

where $F(t|X_{jt})$ is the probability that project j has an release before time t. X_{jt} include the number of commits, contributors and issues that project j has in total up till time t. Notice that the basic Cox regression model does not control for unobserved heterogeneity. One variation of Cox regression including unobserved heterogeneity is to assume the unobserved heterogeneity u_j is strictly exogenous, so one could integrate out the heterogeneity conditional on state variable X_{jt}. However, this setting does not account for endogenous unobserved heterogeneity, such as project innate quality under this context.

2.5 Conclusion

In this Chapter, I provide the institution background of *GitHub*, one of the largest online open source development platform, and the data source of this study. Specifically, I focus on the commit histories of Python projects on the platform. I found that over

90% out of 41529 authors contribute to at most one project per month. These authors are responsible for over 58% of total commits and 79% of total projects in the sample.

After computing the descriptive statistics of the data, it becomes apparent that the lack of manpower is prevalent in the community. Over 90% out of 24345 projects in the data have less than 10 contributors. Even after controlling for projects' ages, the distribution of contribution for projects is still highly skewed. Most projects receive very little contribution while very few projects receive massive amount of contribution.

From the data, we learn that individual developers have limited time and energy for open source development. Moreover, most of their contribution are scattered for many small projects instead of concentrating efforts on bigger projects. This might suggest that contributors would rather be "core developers" in a smaller projects than be "periphery contributors" in bigger projects. From the OLS regression results in Table 2.6 as well as the comparisons in Table 2.9, 2.10, it seems that developers face a trade-off between project popularity and being "core developers".

Chapter 3

A Structural Model of Decentralized Open Source Software Development

3.1 Introduction

In this Chapter, I develop a structural model characterizing individual contributors' preference, and estimate the preference parameters using techniques from dynamic discrete choice models (such as Rust (1987); Hotz and Miller (1993); Arcidiacono and Miller (2011)). Estimating structural models allows us to build a bridge between individual choices in the data and their underlying incentives in the model. The philosophy of this strand of literature is close to the revealed preference theory, that is, individual preferences can be revealed by the choices they make. Compared to "reduced-form" analysis[1], structural models provide a direct theoretical interpretation of the results. In addition, due to the direct link to theoretical model, structural estimations enable the econometricians to measure the effect of counterfactual policies more precisely and in a more tractable way. Therefore, structural models, especially dynamic discrete choice models, are applied to many different fields in economics such as labor economics (Keane and Wolpin, 1997) and industrial organization (Ericson and Pakes, 1995). However, due to the computation burden of estimating a structural model, this approach has not gained as much popularity as reduced-form analysis. Recently, works by Arcidiacono and Miller (2011), Arcidiacono and Miller (2019a) and

[1]Instead of solving for the original theoretically given model form, reduced-form analysis solves dependent variables as equations of the exogenous variables.

Arcidiacono and Miller (2019b) give an alternative approach to estimate and compute counterfactuals in a faster way than the traditional simulation approach using finite dependence property. In this study, I will use the results from Arcidiacono and Miller (2011, 2019a,b) to estimate the model and compute counterfactual analysis.

As argued in previous Chapters, the lack of manpower is prevalent for many open source projects. To understand and potentially mitigate the issue, we need to understand the preferences of individual developers. Motivated by the empirical patterns in Chapter 2, I allow the individual utility to be a flexible function of the author's commits, the chosen project's status, as well as the characteristics of the entire project pool at a given time.

This study departs from the current literature in open source software development in several ways. First, to the extend of my knowledge, this is the first study to build and estimate a dynamic structural model to reveal developer preferences in the open source community. Instead of separating the discussion of developer motivation and open source sustainability, I directly link the developers' incentives with the contribution pattern in the community. In addition, this is the first application for fast estimation and counterfactual computation following Arcidiacono and Miller (2019a) using large-sized data.

3.2 Model

In this section, I will develop a model to characterize the choice for open source developers. It consists of two parts: the choice for contribution by developers, and the "production" outcome of their participation. Each period, an developer will choose the projects to contribute to as well as the effort she puts (measured as the number of commits). The outcome/"production" (if there is a formal release of the project) uses contribution of developers as input, and depends on the project's current status including the historical contribution to the project and the popularity of the project (measured as number of watchers).

Potential externality can arise in the equilibrium because developers might be concerned about their visibility and reputation. For example, if being the owner or core developer of a project brings more reputation to a developer and less costly than

joining other people's project, the developer could prefer to start a small project just to signal her skill. It might not be maintained as soon as the developer finds a new job or get a promotion.

Before the model framework is formally introduced, I will illustrate the basic intuition for the model. Table 3.1 gives an example of the choice problem of an individual developer. This author could choose project A and project B. Project A is a bigger and more popular project with 990 total commits and 100 watchers. Project B is smaller and less popular. Suppose the author wants to contribute 10 commits to one of the project. Her percentage of contribution in project A will be much small than project A. Combining the total watchers, the author would prefer to choose project B even though project B is much less popular since it is much easier to be a core developer in project B.

Table 3.1: An Example of Developer Trade-off

	project A	project B
total watchers	90	10
total commits	990	90
author commits	10	10
% commits	0.001	0.01
% commits × total watchers	0.09	0.1

3.2.1 Developer Project Selection

First we discuss the one-period utility of individual developers if they choose to make commits. Developer i's project choice set at time t is $\mathcal{A}_{it}$ which includes creating her own project from scratch, searching for a new project that she has never contributed to before, or contributing to one of her top 2 past projects ranked by the number of commits she made in the past. Project j has characteristics including the total number of commits, the total number of contributors and the number of watchers/stars at the beginning of the period t. If developer i contribute to project j, she will get a return R at the end of the period. Return R is not observed at the time when the developer is making a choice. But the developer knows that it is a random variable following distribution $F(R; X_{it})$.

The reason of R being random is that the return might depend on the increase in

the number of commits, watchers and authors after t which cannot be observed prior to the author's decision. Here we assume that developer i has a rational belief over the distribution of R, denoted as $F(R; X_{it})$. Empirical distribution $\hat{F}(R; X_{it})$ can be estimated from estimating the distribution of the number of new commits, watchers and authors conditional on the current state variable.

Given choice set $\mathcal{J}_{it}$, developer i is maximizing her payoff over the level of commits in each project. Assume the utility function with return R is $u(R)$ and the cost of commiting to project j is c_j. The cost might include an entry cost if the developer has never contributed to project j before. Thus the developer's utility at a given time period can be written as:

$$u_{ijt}(X_{it}) \equiv E[U(R)|X_{it}] - c_j = \int U(R)dF(R; X_{it}) - c_j$$

The cost of contributing to project j would involve a variable cost which depends on the number of commits, as well as a fixed cost if the developer is creating a new project, or participating in a new project.

The value function for developer i at time t is the maximum of time-discounted accumulation of utility flows. Therefore we can write the agent's dynamic maximization problem as:

$$V_t(X_{it}) = \max_{d_{j\tau}} E\left[\sum_{j\in\mathcal{A}_{it}} \sum_{\tau=t}^{\infty} \beta^{(\tau-t)} d_{ij\tau} \left(u_{ij\tau}(X_{i\tau}) + \varepsilon_{ij\tau}\right)|X_{it}, \varepsilon_{ijt}\right]$$

where X_{it} is the vector of state variables of individual i at time t, $d_{j\tau} = 1$ if the agent choose j at time t and 0 otherwise, $\varepsilon_{ij\tau}$ are *i.i.d.* shocks to utility flow. Note that $\sum_{j\in\mathcal{A}_{it}} d_{ijt} = 1$ for all agent and all time t.

Denote the conditional value function as $v_{jt}(X_{it})$, that is:

$$v_{jt}(X_{it}) = u_{ijt}(X_{it}) + \beta \sum_{x_{t+1}\in\mathcal{X}} V_{t+1}(X_{t+1}) f_{jt}(X_{t+1}|X_t)$$

where $\mathcal{X}$ is the state space, $f_{jt}(X_{t+1}|X_t)$ is the transition probability from state X_t to X_{t+1} conditional on choosing j at time t. The conditional value function is the utility flow without the *i.i.d.* shock ε_{ijt} plus the continuation value. The agent will choose j

if and only if $v_{jt}(X_{it}) + \varepsilon_{ijt} \geq v_{kt}(X_{it}) + \varepsilon_{ikt}$ for every alternative choice k. So we can write the conditional choice probability (CCP) as:

$$P_{jt}(X_{it}) \equiv P(d_{ijt} = 1|X_{it}) = \int \prod_{k \neq j} \mathbb{1}\{\varepsilon_{ikt} - \varepsilon_{ijt} \leq v_{jt}(X_{it}) - v_{kt}(X_{it})\} dF(\varepsilon)$$

Here we assume that the *i.i.d.* shock ε_{ijt} follows Type I Extreme Value distribution.

3.2.2 Production Outcome

A formal release is considered as an important milestone for a project. Therfore in this study, I will consider the production outcome to be whether the project is released or not. Assume the probability for j to have a release at time t to be a function of the project's past commits and authors $(commits_{jt}, authors_{jt})$, as well current commits and authors $(\Delta commits_{jt}, \Delta authors_{jt})$:

$$Pr(release_{jt}|X_{jt}) = f(\Delta commits_{jt}, commits_{jt}, \Delta authors_{jt}, authors_{jt})$$

3.3 Identification and Estimation

3.3.1 Identification of Utility

Agent utility could be identified from choice of contribution as follows. The inversion theorem from Hotz and Miller (1993) shows that for any choice j, there exist a function $\psi_{jt}(\cdot)$ such that

$$\psi_{jt}(P_{jt}(X_{it})) = V_t(X_{it}) - v_{jt}(X_{it})$$

where $V_t(x)$ is the value function at t with state x, and $v_{jt}(x)$ is the conditional value function for choosing choice j, that is,

$$v_{jt}(x_t) = u_{jt}(x_t) + \beta E[V_{t+1}(x_{t+1})|x_t, d_t = j] \tag{3.1}$$

From above we have $v_{jt}(x_t) - v_{0t}(x_t) = \psi_{0t}(x_t) - \psi_{jt}(x_t)$ where $\psi_{jt}(x_t) = \psi(P_{jt}(x_t))$. $P_{jt}(x_t) = P(d_{ijt} = 1|x_t)$ is the probability of choosing choice j conditional on having state variable x_t. Thus, there is a one-to-one mapping between the conditional value

function which is a function of individual utility and conditional choice probability. Normalizing the utility for outside option (not contributing to anything) to zero, the utility of choices $j \neq 0$ are identified.

In addition, we assume that the distribution of exogeneous random utility shocks each period ε_{ijt} follows Type I extreme value distribution. Under this assumption, we have $\psi_{jt}(X_{it}) = \gamma - \ln P_{jt}(x_t)$ where γ is Euler's constant.

3.3.2 Finite Dependence

Following the idea in Arcidiacono and Miller (2011), we can apply finite dependence in this model. Consider the following two choice trajectories:

$$(1)\ d_t = j \Rightarrow d_{t+1} = 0 \Rightarrow d_{t+2} = 0 \Rightarrow d_{t+3} = 0 \Rightarrow d_{t+4} = 0$$
$$(2)\ d_t = 0 \Rightarrow d_{t+1} = j \Rightarrow d_{t+2} = 0 \Rightarrow d_{t+3} = 0 \Rightarrow d_{t+4} = 0$$

Here I argue that two trajectories gives the same distribution of state variable X_{t+5} given initial state X_t.

Table 3.2 gives the results of an OLS regression of the number of current commits (commits made in time t) of project j on its lagged number of commits (number of commits made in time $t-1$, $t-2$, $.etc$). The results show that the correlation between current commits and lagged commits diminishes after 3 periods. Similar for the number of current authors and current watchers, the correlation diminishes after 3 periods as well. Thus, here I argue that the model exhibits 4-period finite dependence.

Following the notations from Arcidiacono and Miller (2011), let $\kappa_\tau(x_{\tau+1}|x_t, j)$ be the distribution of $x_{\tau+1}$ given that the agent takes action j in time t then repeatedly taking the normalizing action $l(x,\tau)$ from $t+1$ to τ. It is recursively defined as:

$$\kappa_\tau(x_{\tau+1}|x_t, j) = \begin{cases} f_{jt}(x_{t+1}|\ x_t) & \text{for } \tau = t \\ \sum_x^X f_{l(x,\tau),\tau}(x_{\tau+1}|x)\kappa_{\tau-1}(x|x_t, j) & \text{for } \tau = t+1, t+2, \cdots \end{cases}$$

In this case of 4-period finite dependence with normalizing action d_0 (not com-

mitting anything), we have:

$$\kappa_\tau(x_{t+2}|x_t, j) = \sum_{x_{t+1}\in\mathcal{X}} f_{0,t+1}(x_{t+2}|x_{t+1})\kappa_t(x_{t+1}|x_t, j)$$
$$\kappa_\tau(x_{t+2}|x_t, 0) = \sum_{x_{t+1}\in\mathcal{X}} f_{j,t+1}(x_{t+2}|x_{t+1})\kappa_t(x_{t+1}|x_t, 0)$$

The conditional value function $v_{jt}(x_t)$ can then be written as:

$$v_{jt}(x_t) = u_{jt}(x_t) + \sum_{\tau=t+1}^{\infty}\sum_{x\in\mathcal{X}} \beta^{\tau-1}[u_{l(x,\tau),\tau}(x_\tau) + \psi_{l(x,\tau),\tau}(x_\tau)]\kappa_\tau(x_{\tau+1}|x_t, j)$$

Using the normalizing action 0 and applying the property of 4-period finite dependence, we have:

$$v_{jt}(x_t) - v_{0t}(x_t) = u_{jt}(x_t) + \sum_{\tau=t+1}^{t+4}\sum_{x_\tau\in\mathcal{X}} \beta^{\tau-1}\left[u_{l(x,\tau),\tau}(x_\tau) + \psi_{l(x,\tau),\tau}(x_\tau)\right] \times \tag{3.2}$$
$$[\kappa_\tau(x_{\tau+1}|x_t, j) - \kappa_\tau(x_{\tau+1}|x_t, 0)] \tag{3.3}$$

Using the inversion theorem (Hotz and Miller, 1993) that $V_t(x) = v_{jt}(x) + \psi_{jt}(x)$, we can re-arrange Equation 3.3 and get the estimation moment equation below:

$$\begin{aligned} u_{jt}(x_t) =& u_{0t}(x_t) + \psi_{0t}(x_t) - \psi_{jt}(x_t) + \\ & \sum_{\tau=t+1}^{t+4}\sum_{x_\tau\in\mathcal{X}} \beta^{\tau-t}\left[u_{0,\tau}(x_\tau) + \psi_{0,\tau}(x_\tau)\right]\left[\kappa_\tau(x_{\tau+1}|x_t, j) - \kappa_\tau(x_{\tau+1}|x_t, 0)\right] \end{aligned}$$

3.3.3 State Space

Before going into the estimation of utility function, I will discuss the state space of agents. In this model, the agent will decide on which project to commit to each period from their choice set. The choice set includes up to 5 choices: not committing to any project (*choice 0*), creating a new project (*choice 1*), searching for a project that the agent has never worked on before (*choice 2*). In addition, if the agent has worked on projects before, she can choose to continue work on one of those old projects. In the data, only 251 authors out of 41529 (0.6%) has contributed to more than 2 projects.

Table 3.2: OLS Regression on Lagged Commits

dep. var	current commits	current authors	current watchers
const	1.368	0.963	0.562
	(0.388)	(0.069)	(0.413)
project commits 1	**0.482**	**0.027**	**0.127**
	(0.039)	(0.008)	(0.036)
project commits 2	**0.077**	**0.007**	**0.051**
	(0.035)	(0.006)	(0.039)
project commits 3	**0.088**	**0.011**	**0.065**
	(0.032)	(0.005)	(0.040)
project commits 4	0.049	0.007	0.022
	(0.032)	(0.005)	(0.034)
project commits 5	0.038	0.001	0.031
	(0.030)	(0.005)	(0.033)
project commits 6	0.060	0.004	0.038
	(0.028)	(0.004)	(0.034)
project commits 7	-0.015	0.000	0.048
	(0.030)	(0.004)	(0.036)
project commits 8	0.041	0.002	0.069
	(0.026)	(0.003)	(0.038)
project commits 9	-0.013	0.001	0.036
	(0.019)	(0.003)	(0.033)
project commits 10	0.046	0.004	0.064
	(0.019)	(0.003)	(0.030)

So here I limited the old projects to be the top 2 projects in terms of the number of commits the agents has made to those projects in the past(*choice 4, 5*).

Given the choice set, the agents will need to make predictions on the future states of the project if they choose to commit to it. The relevant state space would then include: the agent's age (number of months on GitHub platform since registration), the number of projects and commits that the agents has made before. If the agent has contributed to any projects before, the state space will also include the current status of the top 2 projects she worked on in the past, including the project's age, number of commits, authors and watchers up till period t.

Since choice 2 (searching for an existing project on the platform that the agent has never worked on before) depends on the currently available projects on the platform, the state space also contains summary statistics of available projects' status. Specifically, I include the total number of available projects, the inverse of the means of available projects' age, number of total commits, authors and watchers. Figure

2.2 from Chapter 2 shows the trend of the total number of available projects on the platform (divided by 10) as well as the number of new projects by month. The total number of available projects is increasing exponentially through time. I include the inverse of the means of available project status is due to the fact that the empirical distributions of the available projects' status can fit to geometric distribution well[2], and the maximum likelihood estimator for geometric distribution is the inverse of the mean. Figure 3.1 shows the empirical distribution of project age (blue bars) v.s. the fitted geometric distribution (orange curve) from April 2014 to July 2017.

Lastly, I discretize the state space into limited number of bins as there are up to 16 dimensions of the state space. To avoid small sample problem when estimating transition probability and conditional choice probability, I divided each dimension by equal quantiles. There are some dimensions with values concentrated in one value, *e.g.*, over 80% of observations in the author panel data has total past commits to be 0. Therefore I grouped zeros together and separated the rest into 2 bins with equal number of observations (smaller than 5 or greater than 5). In this way, I ended up having 6953 unique bins with 1308083 observations.

3.3.4 Estimation of CCP and Transition

The estimation of conditional choice probability ($P_{jt}(x)$) and transition probabilities ($F(x_{t+1}|x_t)$) follows their statistical analog. That is:

$$\hat{P}_{jt}(x) = \frac{\sum_i^N d_{ijt}\mathbb{1}\{x_{it} = x\}}{\sum_i^N \mathbb{1}\{x_{it} = x\}}$$
$$\hat{f}_j(x_{t+1}|x_t) = \frac{\sum_i^N d_{ijt}\mathbb{1}\{x_{it} = x_t, x_{i,t+1} = x_{t+1}\}}{\sum_i^N d_{ijt}\mathbb{1}\{x_{it} = x_t\}}$$

Due to the curse of dimensionality, even though the data itself is sizeable, the number of observations of each unique state variable combination might not be high enough to satisfy the law of large number. Therefore it could introduce small sample bias to empirical utilities, thus influence the utility estimation results. In addition, the random noise from estimating CCPs and transition empirically might require corrections to the standard deviations of the estimated utility parameters.

[2]The probability mass function of geometric distribution is $P(X = k) = (1-p)^{k-1}p$

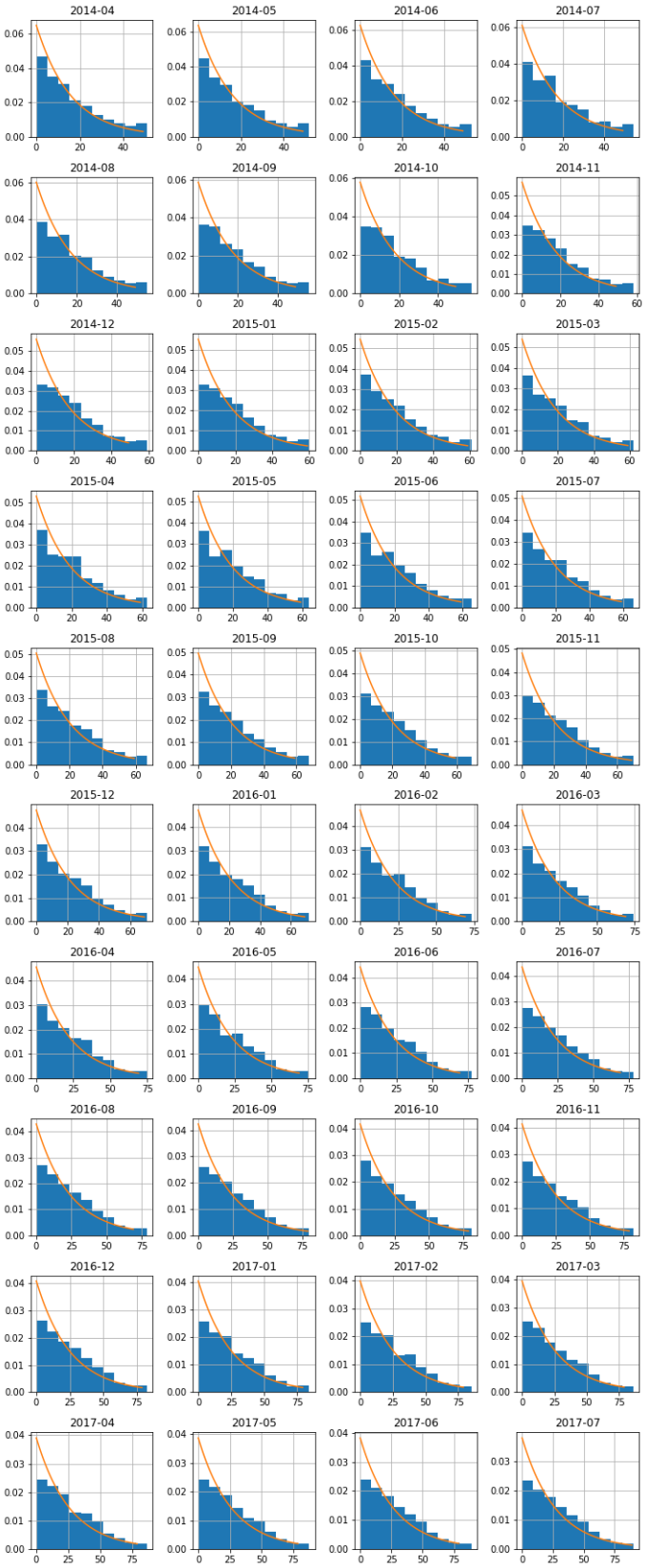

Figure 3.1: Distribution of Project Age Fitted to Geometric Distribution

3.3.5 Estimation of Utility Function

In this section, I will specify the estimation procedure of the utility function. Based on Theorem 3 of Arcidiacono and Miller (2019a) and normalizing action $u_{0t}(x) = 0$, define empirical utility $\hat{u}_{jt}(x)$ as a function of conditional choice probability and transition probability for each choice j, time t and state variable x:

$$\hat{u}_{jt}(x_t) = \psi_{0t}(x_t) - \psi_{jt}(x_t) + \sum_{\tau=t+1}^{t+4} \sum_{x_\tau \in \mathcal{X}} \beta^{\tau-t} \psi_{0\tau}(x_\tau) \left[\kappa_\tau(x_{\tau+1}|x_t, j) - \kappa_\tau(x_{\tau+1}|x_t, 0)\right] \tag{3.4}$$

Based on the inversion theorem of Hotz and Miller (1993), $\psi_{jt}(x)$ is a function of conditional choice probability. Thus, the right-hand-side of the above equation is a combination of conditional choice probabilities ($\psi_{jt}(x) = \gamma - \log P_{jt}(x)$) and transition probabilities $\kappa_\tau(x_{\tau+1}|x_t, j)$).

Here I use a minimum-distance estimator to estimate the paramters (denoted as vector β) of the utility function. That is:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{t=1}^{T} ||u_{jt}(x_{it}; \beta) - \hat{u}_{jt}(x_{it})||^2$$

where $\hat{u}_{jt}(x)$ comes from Equation 3.4 and $u_{jt}(x)$ is the model utility which contains the parameters to be estimated.

Specifically I assume that the utility of choosing option $j \neq 0$ at time t given agent i's state variable to be the following:

$$\begin{aligned}
u_{jt}(x_t) =& E[\beta_0 + \beta_1(\Delta\text{authorcommits}_t + \delta_0\text{authorcommits}_t) + \beta_2(\Delta\text{commits}_t + \delta_1\text{commits}_t) \\
& \beta_3(\Delta\text{authors}_t + \delta_2\text{authors}_t) + \beta_4(\Delta\text{watchers}_t + \delta_3\text{watchers}_t) \mid x] - c_j \\
=& (\beta_0 - c_j) + \beta_1\Delta\text{authorcommits}_t + \beta_1\delta_0\text{authorcommits}_t \\
& + \beta_2 E[\Delta\text{commits}_t|x] + \beta_2\delta_1\text{commits}_t + \beta_3 E[\Delta\text{authors}_t|x] + \beta_3\delta_2\text{authors}_t \\
& + \beta_4 E[\Delta\text{watchers}_t|x] + \beta_4\delta_3\text{watchers}_t
\end{aligned}$$

where state variable contains author's total number of commits made before t, total number of projects participated, their top 2 projects' status, and the current market

conditions.

$$\begin{cases} \Delta\text{authorcommits}_t = \text{author's current number of commits} \\ \text{authorcommits}_t = \text{author's past total commits in the currently chosen projects} \\ \Delta\text{commits}_t = \text{number of new commits of project j at t} \\ \text{commits}_t = \text{total number of past commits of project j before t} \\ \Delta\text{authors}_t = \text{number of new authors of project j at t} \\ \text{authors}_t = \text{total number of past authors of project j before t} \\ \Delta\text{watchers}_t = \text{number of new watchers of project j at t} \\ \text{watchers}_t = \text{total number of past watchers of project j before t} \\ \beta_0 - c_j = \text{base utility for commiting to a project} - \text{cost for commiting to project j} \end{cases}$$

In this model, agents don't know the exact values of new commits, new authors and new watchers at the time when they commit. However, they do know the conditional distribution of new commits, new authors and new watchers. Therefore, they know the expected value of the new commits, new authors and new watchers given their current state.

Since the original state space has high dimensions (up to 16 columns), I discretized the state space by spliting each dimension by quantiles except for the number of projects the agent has participated in the past. For example, the total number of total commits the agent has made in the past has been separated into 0 (the largest bin), between 1 and 5 (the 33% quantile for observations larger than 0), between 5 and 46 (the 67% quantile), and larger than 46. With the discretization, there are 6953 unique bins of the state space.

3.3.6 Estimation Results

To estimate the utility function, we make the following assumptions:

Assumption 1. *The agent is risk neutral.*

Assumption 2. *The dynamic transition is Markov and follows 4-period finite dependence.*

Table 3.3: Summary Statistics of Empirical Utility

choice	count	mean	std	min	25%	50%	75%	max
0	1202696	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	52599	3.49	4.27	-83.94	2.75	3.51	4.35	70.61
2	82857	3.28	5.08	-78.43	2.50	3.09	4.58	54.77
3	177540	-1.01	7.43	-73.57	-0.14	0.51	1.64	67.15
4	3165	-11.70	14.40	-85.96	-23.03	-16.12	0.00	48.24

Note: Choice 0=not committing; Choice 1=creating a new project; Choice 2=searching for a new project that the author has never contributed before; Choice 3/4=commiting to one of the top 2 projects that the author has contributed in the past in terms of the author's number of commits in those projects

Assumption 3. *Contemporary random shocks are i.i.d. across time and choices, and follow Type-I Extreme Value distribution.*

Assumption 4. *Projects are homogeneous conditional on observable variables.*

Given the assumptions above, the minimum distance estimator becomes equivalent to an OLS estimator where the left-hand-side variable is the empirical utility and the right-hand-side is the model utility containing the parameters to be estimated in Equation 3.6.

$$\begin{aligned} u_{ijt}(x) =& (\beta_0 - c_j) + \beta_1(\Delta\text{commits}_{ijt} + \delta_0\text{commits}_{ijt}) \\ &+ \beta_2(E[\Delta\text{commits}_{j,t+1}|x] + \delta_1\text{commits}_{jt}) \qquad (3.5) \\ &+ \beta_3(E[\Delta\text{authors}_{j,t+1}|x] + \delta_2\text{authors}_{jt}) \\ &+ \beta_4(E[\Delta\text{watchers}_{j,t+1}|x] + \delta_3\text{watchers}_{jt}) \qquad (3.6) \end{aligned}$$

Table 3.4 gives the estimation of the utility function with four different specifications. Column (1) gives the baseline estimation where the model utility depends on the the number of commits that the author committed in the current period ("author current commits"), the number of commits that she made to the currently chosen project in the past ("author current project past commits"), the currently chosen project's total number of commits/authors/watchers up till period t ("project past commits/authors/watchers"), the expected number of new commits/authors/watchers that the project gets after period t ("avg project new commits/authors/watchers"), as well as the dummy variables for the category of choices ("choice 1/2/3/4").

Column (2) adds the standard deviations of the currently chosen project's new commits/authors/watchers ("std project new commits/authors/watchers"). The motivation is that the authors might not only care about the mean of the future state of the chosen project, but also the relative range of the future state. Column (3) and (4) adds the square terms and interaction terms in the regression. Column (2) specification gives results not too different from other columns on the common variables, but the adjusted R^2 increases from 0.118 in Column (1) to 0.144. It is also not much different from Column (3) and (4) which have more variables of square terms and interaction terms. Given the estimated parameters of the square and interaction terms are small in scale, I use Column (2) specification as the utility function to be used in the counterfactual analysis.

In Column (2) as well as the other columns, we can see that choice 1 (creating a new project) has the highest parameter ($\beta_0 - c_j$, base-level utility minus the fixed cost for choice j) for the dummy variable, following by choice 2 (searching for a new project that she has not done before), and choice 3 and 4 (continue contributing one of the top 2 past projects). It seems that there is certain intrinsic values in owning a project. Choice 2's high base value might relate to altruism and reciprocity as previous literature indicated. Choice 3 and 4's low base values are related to the fact that there are not many observations with the options of choice 3 and 4 in the choice set. Even with the availability, not many agents actually chose choice 3 or 4. This could indicate that most authors in the data are "casual" contributors who would like to work on new projects rather than continue working on old projects. Thus, there needs to be a positive enough random shock to choice 3 and 4, or negative enough shock to choice 1 and 2 in order for the agent to choose the top 2 past projects instead of creating one's own project or searching for a new project.

Additionally, the agent's utility decreases with the number of current commits the author makes. This might be due to the fact that making commits requires time and effort. Given that the square terms and interaction terms related to the author's current commits are close to zero economically, the negative effect dominates. This factor would be interpreted as the variable cost for making commits.

As for the number of total commits that the project has until period t, the esti-

mated parameter is positive even though it's small in scale. Meanwhile the estimated parameter for the project's past authors is negative, and the parameter for the project's past watchers is positive. So the agent's utility increases when she participates in a popular project, and decreases when there are more competition from her peers.

In addition, the estimated parameters for the project's new states show that agent's utility decreases with the expected number of new commits, while increases with the standard deviation of the number of new commits. It is consistent with the idea that agents prefer to have less dilution of their contribution to the project. Also, the agent's utility increases with the expected number of new authors of the chosen project conditional on the number of new commits and other variables. The number of new authors could be deemed as another measure of popularity, therefore consistent with the fact that agents prefers their chosen project to be more popular. Similarly, agent's utility increases with the expected number of new watchers, which is also consistent with the hypothesis that authors prefer to join a more popular project for the sake of public visibility. Motivated by reputation and career concern, authors would prefer to be a "core" developer in a popular project.

Table 3.4: Utility Estimation Results

	Dependent variable:			
	empirical utility			
	(1)	(2)	(3)	(4)
dummy: create new project	4.356***	4.040***	4.023***	4.010***
	(0.051)	(0.053)	(0.053)	(0.054)
dummy: search new project	3.577***	1.893***	1.867***	1.848***
	(0.049)	(0.058)	(0.058)	(0.058)
dummy: top 1 past project	−0.977***	−1.398***	−1.338***	−1.305***
	(0.030)	(0.031)	(0.032)	(0.033)
dummy: top 2 past project	−12.215***	−11.160***	−11.135***	−11.121***
	(0.161)	(0.159)	(0.159)	(0.160)
author current commits	−0.013***	−0.008***	−0.008***	−0.007***
	(0.001)	(0.001)	(0.001)	(0.002)
author past commits in current project	0.0001	−0.001***	−0.003***	−0.004***
	(0.0001)	(0.0001)	(0.0002)	(0.0002)
project past commits	0.0003***	0.0002***	0.0001***	0.001***
	(0.00003)	(0.00003)	(0.00004)	(0.0001)
project past authors	−0.002***	−0.003***	−0.002***	−0.005***
	(0.0004)	(0.0004)	(0.0004)	(0.001)
project past watchers	0.0001**	0.00002	0.00002	−0.0001**
	(0.00002)	(0.00002)	(0.00003)	(0.00005)
avg project new commits	−0.031***	−0.131***	−0.129***	−0.129***
	(0.002)	(0.002)	(0.002)	(0.002)
std project new commits		0.086***	0.087***	0.088***
		(0.002)	(0.002)	(0.002)
avg project new authors	0.125***	0.185***	0.169***	0.173***
	(0.011)	(0.014)	(0.014)	(0.014)
std project new authors		0.083***	0.087***	0.089***

		(0.014)	(0.014)	(0.014)
avg project new watchers	0.004^{***}	0.005^{***}	0.005^{***}	0.005^{***}
	(0.001)	(0.001)	(0.001)	(0.001)
std project new watchers		-0.007^{***}	-0.007^{***}	-0.007^{***}
		(0.001)	(0.001)	(0.001)
author current commits×			0.00000	0.00000^{*}
project past commits			(0.00001)	(0.00001)
author current commits×			0.00001	0.00000
project past authors			(0.00002)	(0.00002)
author current commits×			-0.00001^{***}	-0.00001^{***}
project past watchers			(0.00001)	(0.00001)
author current project past commits×			0.00000^{***}	-0.00000
project past commits			(0.00001)	(0.00001)
author current project past commits×			-0.00001^{***}	0.00000^{**}
project past authors			(0.00001)	(0.00001)
author current commits^2				-0.00000
				(0.00001)
author current project past commits^2				0.00000^{***}
				(0.00001)
project past commits^2				-0.00000^{***}
				(0.00001)
project past authors^2				0.00000^{***}
				(0.00001)
project past watchers^2				0.000^{***}
				(0.00001)
Observations	182,625	182,625	182,625	182,625

R^2	0.118	0.144	0.144	0.145
Adjusted R^2	0.118	0.144	0.144	0.145
Residual Std. Error	7.414	7.303	7.300	7.299
Degree of Freedom	(182613)	(182610)	(182605)	(182600)
F Statistic	2,027.912***	2,046.286***	1,541.739***	1,237.395***
Degree of Freedom	(12;	(15;	(20;	(25;
	182613)	182610)	182605)	182600)

Note: Choice 0 ommited. *p<0.1; **p<0.05; ***p<0.01

3.3.7 Estimation of Project Production

In this study, I'm interested in counterfactuals that could alter the probability of a project being released, which is a function of the number of new authors $\Delta authors_{jt}$, the number of new commits $\Delta commits_{jt}$ and the number of past authors and commits. The number of new authors $\Delta authors_{jt}$, the number of new commits $\Delta commits_{jt}$ can be estimated from the counterfactual CCPs by integrating over the state space, *e.g.*, $\Delta authors_{jt} = \int \widetilde{P}_{jt}(X_t) dF(X_{it}) \times N_t$.

Here I estimate the hazard rate of releasing a project by using Cox proportional hazard model:

$$P(release|X_{jt}) = f(\Delta commits_{jt}, \Delta authors_{jt}, commits_{jt}, authors_{jt})$$

Potential limits to this method:

- The regression could have endogeneity bias. For example, the agents might be able to identify heterogeneous project qualities that the we cannot observe in data. Then the unobserved qualities would affect both the number of new authors and the probability of release.
- General equilibrium effect is unknown. By changing the CCPs, the market conditions might change.

Table 3.5: Cox Proportional Hazard Regression

	Dependent variable:		
	project age		
	(1)	(2)	(3)
project current commits	0.012^{***}	0.034^{***}	0.034^{***}
	(0.001)	(0.002)	(0.002)
project current authors	0.967^{***}	4.395^{***}	4.164^{***}
	(0.012)	(0.070)	(0.065)
project past commits	0.001^{***}	-0.00004	-0.00002
	(0.0003)	(0.001)	(0.001)
project past authors	-0.172^{***}	-0.144^{***}	-0.234^{***}
	(0.003)	(0.010)	(0.010)
project current commits^2		-0.0002^{***}	-0.0002^{***}
		(0.00002)	(0.00002)
project current authors^2		-0.933^{***}	-0.859^{***}
		(0.024)	(0.022)
project past commits^2		-0.00000	-0.00000
		(0.00000)	(0.00000)
project past authors^2		0.0002^{***}	-0.0001
		(0.0001)	(0.0002)
project current commits$\times$			0.0004^{***}
project past authors			(0.0001)
project current authors$\times$			0.046^{***}
project past authors			(0.004)
Observations	23,729	23,729	23,729
R^2	0.191	0.378	0.376

Max. Possible R^2	0.919	0.919	0.919
Log Likelihood	−27,294.340	−24,174.490	−24,223.850
Wald Test	12,166.580***	10,019.980***	10,230.470***
	(df = 4)	(df = 8)	(df = 10)
LR Test	5,042.185***	11,281.880***	11,183.170***
	(df = 4)	(df = 8)	(df = 10)
Score (Logrank) Test	11,927.140***	19,749.160***	21,187.550***
	(df = 4)	(df = 8)	(df = 10)

Note: *p<0.1; **p<0.05; ***p<0.01

3.4 Counterfactual Analysis

Even though one cannot answer normative questions such as which projects should survive longer to maximize "social utility" as it is hard to distinguish the reason why certain projects are inactive (*e.g.* feature complete, maintainer lost interest), one can still explore potential changes in the distribution of project characteristics in equilibrium.

Based on Theorem 5 of (Arcidiacono and Miller, 2019a), given any temporary payoff innovation where $\Delta_{jt}(x) \equiv \tilde{u}_{jt}(x) - u_{jt}(x) = 0$ for all $t > S$, the counterfactual CCPs for $t < S$ can be identified as follows:

$$\begin{aligned}
\widetilde{\psi}_{jt}(x) - \widetilde{\psi}_{0t}(x) =& \psi_{jt}(x) - \psi_{0t}(x) + \Delta_{jt}(x) - \Delta_{0t}(x) \\
& \sum_{\tau=t+1}^{T} \sum_{x_\tau \in \mathcal{X}} \beta^{\tau-t} [\Delta_{0\tau}(x_\tau) + \widetilde{\psi}_{0\tau}(x_\tau) - \psi_{0\tau}(x_\tau)][\kappa_{\tau-1}(x_\tau|x,j) - \kappa_{\tau-1}(x_\tau|x,0)] \\
& (\rho - \text{period finite dependence}) \\
=& \psi_{jt}(x) - \psi_{0t}(x) + \Delta_{jt}(x) - \Delta_{0t}(x) + \\
& \sum_{\tau=t+1}^{t+\rho} \sum_{x_\tau \in \mathcal{X}} \beta^{\tau-t} [\Delta_{0\tau}(x_\tau) + \widetilde{\psi}_{0\tau}(x_\tau) - \psi_{0\tau}(x_\tau)][\kappa_{\tau-1}(x_\tau|x,j) - \kappa_{\tau-1}(x_\tau|x,0)]
\end{aligned} \tag{3.7}$$

Suppose the regime is temporary, meaning $\widetilde{V}_s(x) = V_s(x)$ for $s > S$. Then we can

solve for the CCPs backwards. At the last period of the payoff innovation regime S, the new CCPs at period $t = S$ can be solved using the following for all choice $j \neq 0$ and all state x:

$$\widetilde{\psi}_{jS}(x) - \widetilde{\psi}_{0S}(x) = \psi_{jS}(x) - \psi_{0S}(x) + \Delta_{jS}(x) - \Delta_{0S}(x)$$

Along with the condition that $\sum_{j=0}^{J} P_{jS}(x) = 1$ for each state x, we can solve $J \times X$ CCPs with $J \times X$ equations where X is the number of states. Moving back one period, we can plug in the new CCPs from $t = S$ into the "continuation term" in Equation 3.7 and solve for CCPs in $t = S-1$. In the following part, I will solve for the counterfactual conditional choice probabilities using the process above.

Increasing Expected Number of Watchers

Here I experiment with the temporary counterfactual where the expected number of watchers rise by 20% for each state and choice from Jan 2015 to Dec 2015, except for choice 1 (creating a new project from scratch). I conduct this temporary counterfactual analysis because we can expect a relative stationary environment in the short-term. After Dec 2015 when the counterfactual period ends, we expect the CCPs to be back to before. This counterfactual scenario might be implemented by promoting projects that are not just created (project age bigger than 1 month) on the platform frontpage, or outside the platform through advertising.

Table 3.6 gives the changes in the *unweighted average* of conditional choice probabilities (average across available states in each month). Since the average changes in CCPs are not weighted by the number of observations in each state, I calculated the expected number of authors that would increase/decrease for each choice in each month t by multiplying the number of authors in state x and time t, and summing up across states in month t. Table 3.6 shows that the number of authors choosing not to commit drops significantly. In addition, the increase in the number of authors choosing to create a new project from scratch is lower than the increase in the number of authors choosing to search for an existing project.

Table 3.6: Change in CCP Under Temporary Counterfactual

month	$E[\Delta P_0(x)]$	$E[\Delta P_1(x)]$	$E[\Delta P_2(x)]$	$E[\Delta P_3(x)]$	$E[\Delta P_4(x)]$
2015-01	-0.34	0.14	0.14	-0.06	0.12
2015-02	-0.36	0.14	0.14	-0.04	0.12
2015-03	-0.33	0.14	0.14	-0.07	0.12
2015-04	-0.36	0.14	0.14	-0.06	0.14
2015-05	-0.35	0.14	0.14	-0.07	0.13
2015-06	-0.36	0.14	0.14	-0.06	0.14
2015-07	-0.33	0.14	0.14	-0.08	0.13
2015-08	-0.35	0.14	0.15	-0.07	0.14
2015-09	-0.36	0.14	0.13	-0.06	0.14
2015-10	-0.34	0.14	0.13	-0.06	0.13
2015-11	-0.38	0.14	0.13	-0.02	0.13
2015-12	-0.36	0.14	0.14	-0.05	0.13

Note: This counterfactual corresponds to the changes in CCP by increasing the expected number of watchers by 20% for each state and choice, except for choice 1 (starting a new project from scratch), starting from Jan 2015 and ending on Dec 2015.

Table 3.7: Change in Number of Authors Under Temporary Counterfactual

month	do nothing	create new project	search new project	top 1 past project	top 2 past project
2015-01	-11530	3203	3539	2012	2553
2015-02	-11534	3130	3517	2053	2592
2015-03	-11615	2941	3478	2163	2759
2015-04	-11382	2965	3343	2122	2707
2015-05	-11445	2812	3338	2226	2799
2015-06	-11374	2793	3230	2246	2827
2015-07	-11199	2730	3110	2208	2839
2015-08	-11083	2649	3104	2190	2848
2015-09	-10969	2597	3019	2201	2849
2015-10	-10815	2558	2913	2202	2842
2015-11	-10532	2611	2795	2108	2729
2015-12	-10416	2299	2675	2231	2926

Note: This counterfactual corresponds to changes in the number of authors with each choices by increasing the expected number of watchers by 20% for each state and choice, except for choice 1 (starting a new project from scratch), starting from Jan 2015 and ending on Dec 2015. The numbers come from summing up the expected number of authors choosing a specific option (number of authors with the same state variable times the new CCP). Negative number indicates that the number of authors choosing that option decreases after the counterfactual policy.

3.5 Limitations and Future Works

In this section, I will discuss the limitations of this study and lay out some future directions of the research. Firstly, this study focuses on Python projects on GitHub. Specifically, I select projects that have "Python" and "library"/"module" in the description in order to exclude projects that are not intended to be released as Python packages such as course materials and personal websites. In this way, I filtered out many projects that could be Python packages as well. Indeed, the total number of Python projects in this study is less than the number of Python packages that can be matched in the Python Package Index, or PyPI based on the study by Valiev et al. (2018). I didn't start from released Python packages in PyPI because the analysis might suffer from selection bias, for example, released packages have better (unobserved) quality than the average project. However, in the future research, one could combine the current dataset with PyPI dataset, and adjust for the imbalance between released projects and other projects.

In addition, it might be worthwhile to collect data on projects of other languages besides Python, such as C, Java, *.etc.* The reason is that we can see a fuller picture on the time and effort allocation of open source contributors.

Apart from the project selection described as above, I also restricted the authors in the sample to be those who contribute to one project at most. In this way, I can restrict author's choice set thus making computation analysis easier. However, we might have selected the "casual" authors instead of the more productive "core" authors. Alternatively, one could model the choice set as a combination of all available projects.

In terms of the demand of open source development, here I used the number of watchers/stars as a proxy to indicate the popularity of open source projects. Since the number of watchers and stars come from the users on GitHub, who may have more expertise of open source development than others, this measure might not fully reflect the usage of general public. If we focus on the released packages on PyPI, we can get the download counts of the projects. One might argue that download counts might be more suitable to represent the utility of general public, thus enabling us to characterize and model the demand side of the open source market.

Given the demand of open source softwares, we can discuss the social welfare of

open source development. In particular, how should a social planner allocate the time and effort of contributors in order to maximize social utility? Should the most contributors focus on projects with potentially high download count, or should contributors scatter the effort in a more uniform manner across different projects?

Another part of relevant information that we could obtain for released projects is the dependency network among packages. The dependency network gives information on which packages (upstream) are used in the source code of a package (downstream). Thus, one can establish an input-output matrix as in the international trade literature to characterize the production of open source packages.

Lastly, on the growth of open source development, an important question here is: can the market evolve to an ergodic state? If so, what would be the properties of the state? With developers sorting and selecting projects through commits, watch and stars, what would be the properties of surviving packages in the long-term? How would potential externalities of open source participation affect the long-term state of the market? In addition, the open source community seems to be in a nonstationary state as the number of new authors and new projects are constantly expanding currently. Suppose there is an aggregate shock to the community, for example, if proprietary companies enforce much stricter contracts with programmers to discourage their participation in open source softwares, then what would happen to the market equilibrium? Future researches might address those questions and investigate the long-term state of the open source community.

3.6 Conclusion

In this Chapter, I build a dynamic discrete choice model to formally characterize the preferences of individual developers, motivated by the empirical patterns presented in Chapter 2. Specifically, I let the individual developer's utility to be a function of their current commits, the chosen project's current status, as well the chosen project's expected future status.

Based on the results of (Arcidiacono and Miller, 2011, 2019b,a), I can identify and estimate the dynamic model using finite-dependence property. The estimation shows that individual developer's utility decreases with the number of current commits, the

chosen project's number of authors in the past, and the expected number of commits in the next period. Moreover, the utility increases with the chosen project's number of commits and number of watchers in the past, as well as the expected number of authors and watchers in the next period. To the extend of my knowledge, this is the first attempt of using dynamic discrete choice model to characterize individual developer preferences.

Lastly, I conduct counterfactual experiment by increasing the expected number of watchers by 20% for all projects except for new projects that are just created from Jan 2015 to Dec 2015. In the counterfactual analysis, I find that the number of authors choosing not to commit drops significantly. Also, the increase in the number of authors choosing to create a new project from scratch is smaller than the increase in the number of authors choosing to search for an existing project during the year of the temporary counterfactual regime.

Bibliography

Arcidiacono, P. and Miller, R. A. (2011). Conditional choice probability estimation of dynamic discrete choice models with unobserved heterogeneity. *Econometrica*, 79(6):1823–1867.

Arcidiacono, P. and Miller, R. A. (2019a). Identifying dynamic discrete choice models off short panels. *Journal of Econometrics*.

Arcidiacono, P. and Miller, R. A. (2019b). Nonstationary dynamic models with finite dependence. *Quantitative Economics*, 10(3):853–890.

Athey, S. and Ellison, G. (2014). Dynamics of open source movements. *Journal of Economics & Management Strategy*, 23(2):294–316.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the Factors That Impact the Popularity of GitHub Repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344. IEEE.

Borges, H. and Tulio Valente, M. (2018). What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software*, 146:112–129.

Coase, R. H. (1937). The Nature of the Firm. *Economica*, 4(16):386–405.

Coelho, J. and Valente, M. T. (2017). Why modern open source projects fail. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, pages 186–196, New York, New York, USA. ACM Press.

Demsetz, H. (1967). Toward a Theory of Property Rights. *The American Economic Review*, 57(2):347–359.

Eghbal, N. (2016). Roads and bridges: the unseen labor behind our digital infrastructure.

Ericson, R. and Pakes, A. (1995). Markov-perfect industry dynamics: A framework for empirical work. *The Review of economic studies*, 62(1):53–82.

Foucault, M., Palyart, M., Blanc, X., Murphy, G. C., and Falleri, J.-R. (2015). Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 829–841.

Fronchetti, F., Wiese, I., Pinto, G., and Steinmacher, I. (2019). What attracts newcomers to onboard on oss projects? tl; dr: Popularity. In *IFIP International Conference on Open Source Systems*, pages 91–103. Springer.

Gächter, S., von Krogh, G., and Haefliger, S. (2010). Initiating private-collective innovation: The fragility of knowledge sharing. *Research Policy*, 39(7):893–906.

Geiger, R. S. (2017). Summary analysis of the 2017 github open source survey. *arXiv preprint arXiv:1706.02777*.

Georgios Gousios (2013). The GHTorent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, page 438. IEEE Press.

Hargrave, T. J. and Van De Ven, A. H. (2006). A Collective Action Model of Institutional Innovation. *Academy of Management Review*, 31(4):864–888.

Hotz, V. J. and Miller, R. A. (1993). Conditional choice probabilities and the estimation of dynamic models. *The Review of Economic Studies*, 60(3):497–529.

Johnson, J. P. (2002). Open source software: Private provision of a public good. *Journal of Economics & Management Strategy*, 11(4):637–662.

Keane, M. P. and Wolpin, K. I. (1997). The career decisions of young men. *Journal of political Economy*, 105(3):473–522.

Khondhu, J., Capiluppi, A., and Stol, K.-J. (2013). Is It All Lost? A Study of Inactive Open Source Projects. pages 61–79. Springer, Berlin, Heidelberg.

Lakhani, K. and Wolf, R. G. (2003). Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. *SSRN Electronic Journal*.

Langlois, R. N. and Garzarelli, G. (2008). Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration. *Industry and Innovation*, 15(2):125–143.

Lerner, J. and Tirole, J. (2003). Some Simple Economics of Open Source. *The Journal of Industrial Economics*, 50(2):197–234.

Miller, C., Widder, D. G., Kästner, C., and Vasilescu, B. (2019). Why do people give up flossing? a study of contributor disengagement in open source. In *IFIP International Conference on Open Source Systems*, pages 116–129. Springer.

Qiu, H. S., Li, Y. L., Padala, S., Sarma, A., and Vasilescu, B. (2019). The signals that potential contributors look for when choosing open-source projects. In *ACM Conference on Computer-Supported Cooperative Work and Social Computing*, CSCW. ACM.

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

Rust, J. (1987). Optimal replacement of gmc bus engines: An empirical model of harold zurcher. *Econometrica: Journal of the Econometric Society*, pages 999–1033.

Shah, S. K. (2004). Understanding the nature of participation & coordination in open and gated ssource software development communities. In *Academy of Management Proceedings*, volume 2004, pages B1–B5. Academy of Management Briarcliff Manor, NY 10510.

Shah, S. K. (2006). Motivation, governance, and the viability of hybrid forms in open source software development. *Management science*, 52(7):1000–1014.

Valiev, M., Vasilescu, B., and Herbsleb, J. (2018). Ecosystem-level determinants of sustained activity in open-source projects: a case study of the PyPI ecosystem.

In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 644–655, New York, New York, USA. ACM Press.

von Krogh, G., Haefliger, S., Spaeth, S., and Wallin, M. W. (2012). Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, 36(2):649–676.

von Krogh, G., Spaeth, S., and Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241.

Xu, L., Nian, T., and Cabral, L. (2019). What makes geeks tick? a study of stack overflow careers. *Management Science*.

Appendices

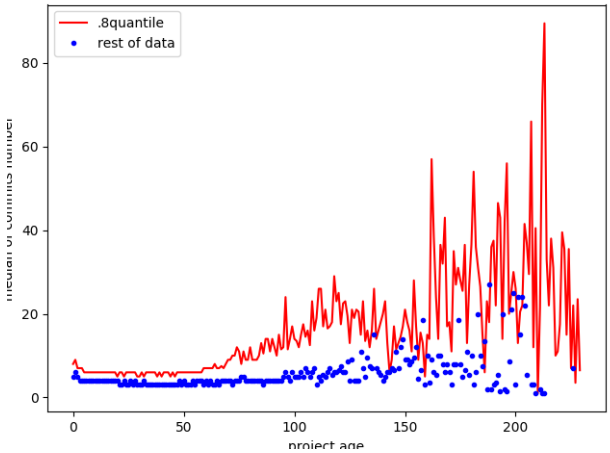

Figure 2: Average Number of Commits by Python Project Age

Table 8: Summary Statistics From Original Data

	mean	s.d.	median	(max, min)	no. obs
monthly commits per author	15.49	314.54	4	(656243, 1)	37990105
monthly projects per author	2.04	17.67	1	(52435, 0)	37990105
total commits per project	22.50	827.24	4	(2332481, 1)	20272825
total contributors per project	1.48	12.03	1	(40512, 1)	23095862
project age	22.94	16.09	18	(116, 5)	30077852
date of last commit	20.29	13.57	16	(234, 1)	25959769
issues per project	10.51	174.35	1	(145558, 1)	4079030
pull requests per project	7.19	96.04	1	(54805, 1)	2817013
watchers per project	13.31	215.73	1	(144323, 1)	4786959

Note: Monthly commits and projects include projects that are forked from other projects. Number of authors per project excludes forked projects. "Date of last commit" refers to the month when the last commit to a certain project was pushed. For example, date of last commit of 6 means that the last commit of the project was pushed 6 months before the current observation date (July 2017). In addition, it eliminated the commits that are before Jan 1998 (git was invented) and after July 2017. Those commit dates are due to random user system errors such as dead CMOS batteries.

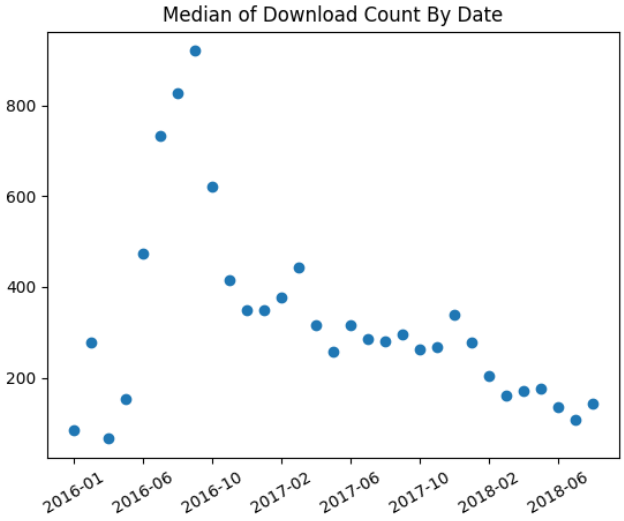

Figure 3: Median of Download Count by Month

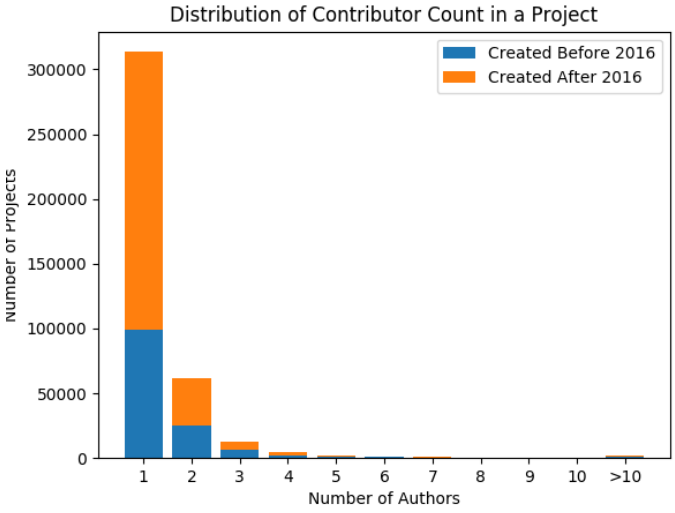

Figure 4: Distribution of Project Size

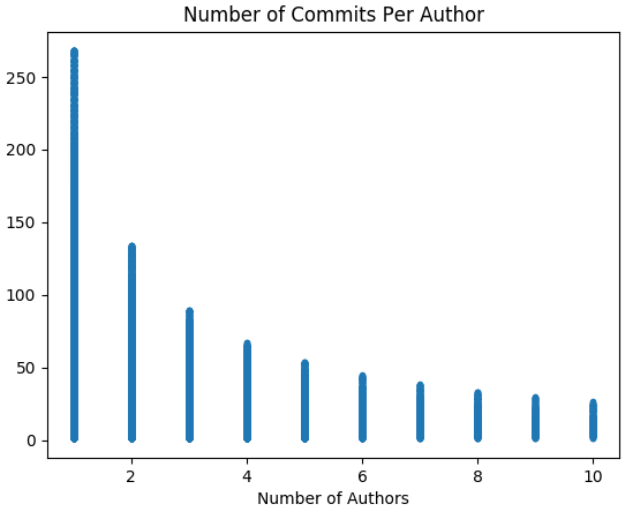

Figure 5: Productivity Per Author By Project Size

Table 9: Panel OLS Regression of Project Popularity

	baseline	org panel	ind panel	org panel	ind panel
num commits	0.1385	0.0220	0.0690	0.0185	0.0695
s.e.	0.0075	0.0138	0.0096	0.0139	0.0096
num commits sq	0.0000	0.0000	-0.0000	0.0000	-0.0000
s.e.	0.0000	0.0000	0.0000	0.0000	0.0000
num authors	5.7712	5.6512	27.5644	5.4604	27.5764
s.e.	0.4267	0.4626	1.2079	0.4618	1.2191
num authors sq	-0.0016	-0.0072	-0.0774	-0.0070	-0.0775
s.e.	0.0013	0.0014	0.0035	0.0013	0.0035
project f.e.	N	Y	Y	Y	Y
age f.e.	N	N	N	Y	Y
no. obs	6622959	487587	6135372	487587	6135372

Note: "org panel" and "ind panel" refers to panel OLS estimation for organization owned projects and individual owned projects respectively.

Table 10: Python Community Contributor Specialization

	mean	s.d.	median	(max, min)	no. obs
monthly commits in Python	8.62	23.79	3	(11532, 1)	1063238
monthly total commits	37.35	677.20	10	(496724, 1)	1063238
Python commit proportion	0.58	0.39	0.59	(1, 0)	1063238
monthly Python projects	1.17	1.07	1	(201, 1)	1063238
monthly total projects	4.71	52.90	2	(16809, 1)	1063238
Python project proportion	0.58	0.35	0.50	(1, 0)	1063238

Note: Number of observation is the number of developers times the number of months. Here the python projects are selected such that the project description contains "Python" and the creation date is after Jan 2012.

Table 11: Python Projects and Releases

year	new projects	organization owned	releases	avg release year
2012	7798	978 (12.54%)	1203 (15.43%)	2012.51
2013	11340	1449 (12.78%)	1738 (15.33%)	2013.37
2014	14498	2086 (14.39%)	2219 (15.31%)	2014.29
2015	19594	2655 (13.55%)	2840 (14.49%)	2015.13
2016	20331	2876 (14.15%)	2507 (12.33%)	2016.08
2017	23040	2595 (11.26%)	2260 (9.81%)	2016.97
year	released by org	release by individual	released watchers	unreleased watchers
2012	270 (27.61%)	933 (13.68%)	135.28	23.93
2013	353 (24.36%)	1385 (14.0%)	85.54	25.69
2014	525 (25.17%)	1694 (13.65%)	77.38	22.73
2015	680 (25.61%)	2160 (12.75%)	50.24	16.25
2016	622 (21.63%)	1885 (10.8%)	29.27	11.19
2017	485 (18.69%)	1775 (8.68%)	38.25	9.45

Note: The Python projects are selected from GitHub that are created between Jan 2012 and Dec 2017 (96601 projects in total), and whose project description contains "Python" and does not contain "tutorial", "learn", "study", "example", "school", "exercise", "course", "blog", "template".

Table 12: Python Projects and Releases

year	new projects	org. owned	releases	avg release year	released by org	release by ind
1998	6	3	1	2007	0 (0%)	1 (33%)
1999	13	3	1	2013	1 (33.0%)	0 (0%)
2000	16	5	4	2011.25	2 (40%)	2 (18%)
2001	13	4	1	2015	1 (25%)	0 (0%)
2002	31	8	2	2010.5	1 (12%)	1 (4%)
2003	23	4	11	2012.55	3 (75%)	8 (42%)
2004	18	0	3	2010.67	0	3 (17%)
2005	50	16	11	2009	7 (44%)	4 (12%)
2006	88	32	13	2010.08	7 (22%)	6 (11%)
2007	186	67	26	2011.08	13 (19%)	13 (11%)
2008	353	88	66	2010.76	22 (25%)	44 (17%)
2009	931	134	161	2011.05	33 (25%)	128 (16%)
2010	1674	259	357	2011.14	65 (25%)	292 (21%)
2011	3530	635	678	2011.92	188 (30%)	490 (17%)
2012	9017	1055	1216	2012.69	261 (25%)	955 (12%)
2013	20652	1897	1949	2013.49	387 (20%)	1562 (8%)
2014	32542	3180	2588	2014.37	578 (18%)	2010 (7%)
2015	57182	4655	3605	2015.21	822 (18%)	2783 (5%)
2016	81677	5772	3462	2016.12	797 (14%)	2665 (4%)
2017	116272	6240	3502	2016.99	699 (11%)	2802 (3%)

Note: The Python projects are selected from GitHub that are created before 2018 (324274 projects in total), and whose project description contains "Python" and does not contain "tutorial", "learn", "study", "example", "school", "exercise", "course", "blog", "template", "interview" and "practice".

NOTE: some Python projects in the sample are not intended to be released as a Pypi-package, such as `https://github.com/robbyrussell/oh-my-zsh` and `https://github.com/vinta/awesome-python`.

Table 13: Python Projects At Releases By Creation Cohort (2010-2017)

		Commits		Authors	
year	Age at Release	(release)	(unreleased)	(release)	(unreleased)
2010	405.59	89.38	88.13	9.74	3.44
2010	69	4	12	2	2
2011	328.1	115.57	87.84	14.46	2.91
2011	64	26	11	8	1
2012	244.74	172.75	46.61	21.54	2.15
2012	43	124	8	22	1
2013	176.91	260.23	39.23	24.83	1.85
2013	12	220	7	22	1
2014	137.67	355.18	30.54	28.89	1.6
2014	11	321	7	26	1
2015	79.76	471.7	23.11	37.8	1.44
2015	7	454	6	38	1
2016	48.5	597.0	15.88	47.38	1.3
2016	4	639	5	49	1
2017	11.44	676.83	13.07	55.52	1.23
2017	1	681	5	57	1

Note: Here "year" refers to the year of a project's creation. The first row for a given year shows the average of the value and the second row shows the median. "Release age" refers to the average project's age in days when it is first released. "Commits (release)" and "authors (release)" refer to the average number of commits and number of authors at the release time of a released project. "Commits (unreleased)" and "authors (unreleased)" refer to the average number of commits and number of authors when an unreleased project is 125 days old, which is the average release age for all released projects.

Table 14: Python Projects At Releases By Creation Cohort (1998-2009)

		Commits		Authors	
year	Age at Release	(release)	(unreleased)	(release)	(unreleased)
1998	3270.0	0.0	15512.6	0.0	75.6
1998	3270	0	6719	0	65
1999	4926.0	218.0	746.5	22.0	8.0
1999	4926	218	120	22	4
2000	4201.0	158.5	2870.0	11.5	19.25
2000	4258	158	180	11	3
2001	5146.0	450.0	2589.92	37.0	64.08
2001	5146	450	114	37	4
2002	3129.0	242.5	365.83	21.0	7.93
2002	3129	242	105	21	4
2003	3514.55	290.27	230.83	23.64	8.92
2003	3934	317	151	23	4
2004	2538.67	107.0	308.6	8.67	5.53
2004	2063	0	51	0	2
2005	1456.0	34.64	563.85	4.18	7.54
2005	1585	0	92	0	3
2006	1485.46	131.31	411.11	13.0	11.24
2006	1007	0	51	0	3
2007	1508.77	180.5	255.16	16.15	7.09
2007	1917	123	33	21	3
2008	985.03	125.38	286.54	11.82	5.42
2008	645	0	29	0	2
2009	731.21	111.35	170.58	11.06	6.51
2009	429	3	18	1	2

Note: Here "year" refers to the year of a project's creation. The first row for a given year shows the average of the value and the second row shows the median. "Release age" refers to the average project's age in days when it is first released. "Commits (release)" and "authors (release)" refer to the average number of commits and number of authors at the release time of a released project. "Commits (unreleased)" and "authors (unreleased)" refer to the average number of commits and number of authors when an unreleased project is 125 days old, which is the average release age for all released projects.

Table 15: Python Projects Release By Popularity

watcher count percentile	age	commits	authors	releases	total
0% - 72%	846.48	12.1	1.25	0.02	227611
(0 - 1)	730.0	4.0	1.0		
72% - 83%	1085.64	27.17	1.46	0.07	51547
(1 - 2)	985.0	9.0	1.0		
83% - 90%	1198.53	41.47	1.63	0.13	26259
(2 - 4)	1096.0	13.0	1.0		
90% - 97%	1352.25	71.88	2.17	0.22	24401
(4 - 25)	1263.0	20.0	1.0		
97% - 99%	1554.38	160.52	3.79	0.36	6481
(25 - 110)	1496.0	47.0	2.0		
99% - 100%	1696.59	568.83	14.41	0.45	3227
(110 - 70266)	1661.0	148.0	5.0		

Note: Total number of watchers quantile: (72%, 1); (83%, 2); (90%, 4); (97%, 25); (99%, 110), (100%, 70266). First row is the mean, second row is median. Age unit is by day.

Table 16: Commits Before and After Topic Creation

	authors per day after	authors per day before	diff
mean	0.0278	0.0137	0.0141
s.d.	0.0303	0.0171	0.0338

Note: GitHub introduced "project topics" feature on Jan 30, 2017. With this feature, project owners can add topics like "machine-learning", "python" to their repository to facilitate repository searches. Also, GitHub pushes "trending repositories" based on project topics each day. Samples are projects that have topics, and project was created at least 30 days before introducing topic, the last commit was made at least 30 days after introducing topic. Sample size is 99 projects.

Table 17: Normalized Utility Estimation

	Dependent variable:			
	empirical utility			
	(1)	(2)	(3)	(4)
choice1	5.517***	5.060***	5.046***	5.037***
	(0.056)	(0.055)	(0.055)	(0.055)
choice2	6.026***	4.981***	4.961***	4.947***
	(0.063)	(0.065)	(0.065)	(0.065)
choice3	5.223***	4.537***	4.555***	4.564***
	(0.075)	(0.074)	(0.074)	(0.074)
author current commits	−0.212***	−0.137***	−0.144***	−0.137***
	(0.018)	(0.018)	(0.020)	(0.024)
author current project past commits	0.022	−0.241***	−0.486***	−0.619***
	(0.020)	(0.020)	(0.030)	(0.035)
project past commits	0.306***	0.248***	0.205***	0.547***
	(0.036)	(0.036)	(0.037)	(0.084)
project past authors	−0.209***	−0.272***	−0.231***	−0.481***
	(0.044)	(0.043)	(0.044)	(0.079)
project past watchers	0.059**	0.024	−0.037	−0.207***
	(0.027)	(0.027)	(0.028)	(0.049)
avg project new commits	−0.646***	−2.724***	−2.687***	−2.691***
	(0.031)	(0.044)	(0.044)	(0.044)
std project new commits		2.502***	2.534***	2.545***
		(0.044)	(0.044)	(0.044)
avg project new authors	0.431***	0.639***	0.578***	0.597***
	(0.037)	(0.048)	(0.049)	(0.049)
std project new authors		0.344***	0.361***	0.373***
		(0.057)	(0.057)	(0.057)
avg project new watchers	0.164***	0.195***	0.212***	0.213***

	(0.026)	(0.038)	(0.038)	(0.038)
std project new watchers		-0.460^{***}	-0.483^{***}	-0.487^{***}
		(0.043)	(0.043)	(0.043)
author current commits×			0.028	0.045^{*}
project past commits			(0.023)	(0.025)
author current commits×			0.016	0.005
project past authors			(0.038)	(0.039)
author current commits×			-0.124^{***}	-0.127^{***}
project past watchers			(0.035)	(0.036)
author current project past commits×			0.149^{***}	-0.036
project past commits			(0.015)	(0.028)
author current project past commits×			-0.146^{***}	0.082^{**}
project past authors			(0.020)	(0.035)
author current commits^2				-0.001
				(0.002)
author current project past commits^2				0.037^{***}
				(0.005)
project past commits^2				-0.034^{***}
				(0.008)
project past authors^2				0.037^{***}
				(0.009)
project past watchers^2				0.015^{***}
				(0.003)
Observations	182,625	182,625	182,625	182,625
R^2	0.117	0.144	0.144	0.145
Adjusted R^2	0.117	0.144	0.144	0.145

Residual Std. Error	7.415 (df = 182614)	7.303 (df = 182611)	7.301 (df = 182606)	7.299 (df = 182601)
F Statistic	2,208.787*** (df = 11; 182614)	2,189.546*** (df = 14; 182611)	1,621.782*** (df = 19; 182606)	1,288.580*** (df = 24; 182601)

Note: Dummy for choice 4 causes matrix singularity after normalization.

*p<0.1; **p<0.05; ***p<0.01

Table 18: Choice-specific Utility Estimation

	Dependent variable:			
	empirical utility			
	(1)	(2)	(3)	(4)
Constant	1.776^{***}	1.925^{***}	-1.162^{***}	-12.666^{***}
	(0.448)	(0.092)	(0.035)	(0.476)
author current commits	0.001	-0.011^{***}	-0.008^{***}	-0.105^{**}
	(0.002)	(0.003)	(0.001)	(0.041)
author current project past commits			-0.002^{***}	0.067^{***}
			(0.0001)	(0.022)
project past commits		0.0001^{**}	0.0002^{***}	-0.003^{***}
		(0.0001)	(0.00004)	(0.001)
project past authors		-0.002^{***}	-0.002^{***}	0.024^{***}
		(0.001)	(0.001)	(0.009)
project past watchers		0.0001^{**}	-0.00001	-0.001^{*}
		(0.00003)	(0.00003)	(0.001)
avg project new commits	-0.247^{***}	-0.198^{***}	-0.114^{***}	-0.014
	(0.008)	(0.006)	(0.002)	(0.022)
std project new commits	0.164^{***}	0.119^{***}	0.065^{***}	0.080
	(0.008)	(0.004)	(0.002)	(0.053)
avg project new authors	-2.544^{***}	0.547^{***}	-0.038^{**}	-0.078
	(0.412)	(0.033)	(0.018)	(0.141)
std project new authors	11.627^{***}	0.081^{***}	0.300^{***}	-0.462
	(0.358)	(0.024)	(0.018)	(0.284)
avg project new watchers	-0.006	-0.002	-0.003^{**}	0.029^{**}
	(0.009)	(0.002)	(0.001)	(0.013)
std project new watchers	0.069^{***}	-0.021^{***}	0.003^{***}	0.022
	(0.003)	(0.001)	(0.001)	(0.017)

Observations	23,184	32,193	124,971	2,157
R^2	0.171	0.068	0.037	0.024
Adjusted R^2	0.171	0.068	0.037	0.019
Residual Std. Error	4.539 (df = 23176)	5.860 (df = 32182)	7.714 (df = 124959)	15.261 (df = 2145)
F Statistic	684.475*** (df = 7; 23176)	235.592*** (df = 10; 32182)	438.577*** (df = 11; 124959)	4.732*** (df = 11; 2145)

Note: *p<0.1; **p<0.05; ***p<0.01

Table 19: Cox Proportional Hazard Regression

	Dependent variable:		
	project age		
	(1)	(2)	(3)
project current commits	0.011^{***}	0.032^{***}	0.032^{***}
	(0.001)	(0.002)	(0.002)
project current authors	0.985^{***}	5.292^{***}	5.373^{***}
	(0.012)	(0.085)	(0.086)
project current watchers	-0.0003	0.0004	0.0004
	(0.0003)	(0.001)	(0.001)
project current commits2		-0.0002^{***}	-0.0002^{***}
		(0.00002)	(0.00002)
project current authors2		-1.349^{***}	-1.413^{***}
		(0.033)	(0.034)
project current watchers2		0.00000	0.00000
		(0.00000)	(0.00000)
project past commits	0.0004	-0.0004	0.0001
	(0.001)	(0.001)	(0.001)
project past authors	-0.115^{***}	-0.166^{***}	-0.261^{***}
	(0.007)	(0.010)	(0.015)
project past watchers	-0.006^{***}	0.003^{***}	-0.002^{*}
	(0.001)	(0.001)	(0.001)
project past commits^2		0.00000^{***}	-0.00000^{***}
		(0.00000)	(0.00000)
project past authors^2		0.0004^{***}	0.0002^{***}
		(0.00003)	(0.0001)
project past watchers^2		-0.00001^{***}	-0.00000
		(0.00000)	(0.00000)
project current commits$\times$			0.0001

project past authors			(0.0003)
project current authors×			0.057***
project past authors			(0.005)
project current commits×			−0.0001***
project past watchers			(0.00002)
project current authors×			0.003***
project past watchers			(0.001)
Observations	23,729	23,729	23,729
R^2	0.195	0.385	0.388
Max. Possible R^2	0.919	0.919	0.919
Log Likelihood	−27,246.380	−24,044.210	−23,994.480
Wald Test	12,544.510***	8,883.050***	8,747.480***
	(df = 6)	(df = 12)	(df = 16)
LR Test	5,138.109***	11,542.440***	11,641.910***
	(df = 6)	(df = 12)	(df = 16)
Score (Logrank) Test	12,032.200***	19,902.110***	21,921.360***
	(df = 6)	(df = 12)	(df = 16)

Note: *p<0.1; **p<0.05; ***p<0.01