An Empirical Study on Open Source Ecosystems of Fortune Global 500 Companies

By Zitong Su

A project report submitted to the Graduate Program in School of Computing in conformity with the requirements for the Degree of Master of Science

Queen's University
Kingston, Ontario, Canada
November, 2021

Abstract

The collaborative and sharing characteristics of Open-Source Software (OSS) attract commercial companies to make their projects open-source and also attract external contributions from developers outside a company. As the number of open-source projects and contributors in a company grows, an open-source ecosystem gradually comes into being. Due to the high financial and time cost of building up an OSS ecosystem from scratch by trial-and-error, there is a demand in characterizing successful ecosystems to provide guidelines for other companies who want to develop their own OSS environment. It helps avoid invalid development efforts and reduces costs. In this report, we focus on Fortune Global 500, the 500 most influential commercial companies worldwide, such as Apple, Amazon, and Microsoft, and analyze over 4 million commit messages from more than 2,000 GitHub repositories. We compare the number of external and internal developers and explore the external participation situation in the ecosystems. Moreover, we offer a statistical view on the commit composition of these OSS ecosystems. We also summarize a few characteristics of the ecosystems. Our study shows that there are more external than internal developers participating in the OSS development process, but external developers have 41.2% percent less commit number in total than internal ones. Three code change composition patterns of perfective-dominant, perfective-leading, and adaptive-dominant are

found. The popular topics, repository age, code change frequency of the ecosystems are characterized by comparing different business sectors, such as technology, retailing, and financials. To improve OSS ecosystem diversity, we appeal to companies to raise the external commit acceptance rate in order to maintain a lively OSS ecosystem. Practitioners from different business sectors could adopt our corresponding analysis result to help construct their own OSS ecosystems for strategic growth.

Acknowledgments

First, I would like to express my gratitude to my supervisor Professor Ying (Jenny) Zou, who has brought me to Queen's and offered me a chance to experience the amazing academic journey here. It is her consistent guidance that helps me make continuous progress in my research project. I would also like to thank her for offering warm help when I arrived in Kingston during the severe pandemic.

Besides, I would like to thank Professor Bram Adams who gave insightful guidelines on choosing the research project topic and followed up with my project progress throughout the project.

Also, I would like to say thank you to Stefanos Georgiou, the postdoctoral fellow in Software Evolution and Analytics Lab, who has helped me label the dataset, provided feedbacks for my project report, and corrected grammar mistakes. I also thank Chunli Yu for helping me with the manual labelling process.

Thanks to all the labmates Guoliang Zhao, Chunli Yu, Dr. Stefanos Georgiou, Dr. Taher Ahmed Ghaleb, Osama Ehsan, Maram Assi, Shayan Noei, Yiping Jia, and Fangjian Lei for creating a friendly and delightful study environment. It is a great memory to meet all of you at the picnic in lake Ontario park.

Last but not least, I would like to express my deepest love to my parents who always give continuous supports, encouragement, and guidance through my graduate

study life.

Life is a journey. Everyone you met makes it colourful and unique. And I am always enjoying the ride.

Contents

List of Tables

List of Figures

Section 1

Introduction

Open-Source Software (OSS) is a paradigm developed out of the proprietary software model [1, 2, 3, 4]. It brings together software developers from all around the world regardless of their geography, language, time zone, and firm boundaries [5, 6]. Such fact increases the working flexibility and efficiency [7] because software practitioners do not need to work under a fixed time schedule, suffer from physical distance limitation, and be restricted to traditional quality management process [8, 9, 10, 11].

As the open-source paradigm has been gradually adopted by more and more developers, commercial companies start to build business models fitting into the open innovation environment [12]. Many companies set up their own OSS ecosystems to attract project contributions through external practitioners [13, 14, 15]. Tech-giants like Google [16], Microsoft [17], and Facebook [18] have built up OSS ecosystems and open sourced some of their products, such as Google's Android, Facebook's React, and Microsoft's ASP.NET framework [19].

Building up an active OSS ecosystem is not an easy task. OSS projects could fail due to many reasons, such as competitor usurpation, functional obsoletion, lack of time and interest of major contributors, and outdated technologies [20]. Although

tech-giants could afford the financial and time cost of trial-and-error, it is not practical for smaller companies to build their own OSS ecosystems from scratch by trial-and-error. There is a need to gain experience from mature OSS ecosystems, summarize characteristics these ecosystems have in common, and set up a list of guidelines to help companies build up ecosystems in a more cost-saving way and avoid invalid efforts.

As mentioned in the motivation part, there is a gap in providing a statistical view on and characterizing influential enterprise-based OSS ecosystems to help small companies build up an OSS ecosystem efficiently. We search through studies related to OSS ecosystems to see if there is any inspiration for filling the gap. Zhang et al. [21] explore company collaborations within OSS ecosystems on OpenStack. They identify different engagement strategies that companies employ for participation in OpenStack and characterize company collaboration patterns. Linåker et al. [22] perform a case study on the Apache Hadoop ecosystem to explore changing stakeholder influence and collaboration patterns. Their findings show that both influence shifting and collaborations exist between rivalling and non-competing firms. Matragkas et al. [23] make an analogy between OSS ecosystems and ecological communities to investigate the diversity and structure of OSS communities. They find that GitHub is made up of core, active, and passive users. The percentage of core developers and active users does not change as the project grows. The existing studies mainly focus on analyzing collaborative patterns and diversity within the OSS ecosystem with limited data sources. However, none of them have analyzed the GitHub ecosystems of Global 500, where many tech-giant companies reside.

To fill the gap mentioned above, our project collects code change information

from GitHub and performs statistical analysis on the OSS ecosystems of Global 500 companies. The major contributions of this project are as follows:

- We study the identities of contributors to understand who are the major driving forces of the Global 500 OSS ecosystems. The result shows that 39.5% of contributors are company employees who contribute to their own company's OSS ecosystem. 60.5% of contributors are external developers other than company employees. External developers only contribute 29.4% of the total number of commits. And *individual* developers have the lowest per capita commit number among external contributor groups. The results of this RQ can be integrated into a company's OSS development strategy and attract more contributions to its OSS ecosystem.

- We perform code change composition analysis on code change commit messages and detect three patterns of code change compositions, including perfective-dominant, perfective-leading, and adaptive-dominant. We find that corrective code change does not occur as the dominant code change type. The results of this RQ give practitioners some understandings on the code change composition patterns in mature OSS ecosystems. Moreover, it provides some guidance to newly-built OSS ecosystems to plan their evolution directions.

- We characterize the OSS ecosystems based on different business sectors and discover four characteristics as follows. Software development life cycle relevant topics are popular trends for the ecosystems. But ecosystems in different business sectors have their unique focus. Ecosystems in all business sectors experience only one major open-source wave, a period during which a large number

of GitHub repositories are created intensively, to become mature. *Technology* sector has repositories with the highest commit growth rate.

The remainder of this report is organized as follows:

Section 2, Related Work: We summarize prior studies related to OSS ecosystems, code change classification, and identity disambiguation.

Section 3, Definition of Terminologies: We give definitions on two commonly used terminologies in the report.

Section 4, Experimental Setup: We describe the overall workflow and data collection process in this section.

Section 5, Experimental Results: Experimental results for each research question are presented in this section.

Section 6, Threats to Validity: The internal and external threats to the validity of our empirical study are discussed in this section.

Section 7, Conclusion: We draw a conclusion and present the works to be done in the future in the last section.

Section 2

Related Work

In this section, we discuss studies related to (1) OSS ecosystems, (2) code change classification, and (3) identity disambiguation.

2.1 OSS Ecosystems

Jansen et al. [24] study the technical and business aspects of software engineering in vibrant ecosystems. The authors point out that failing to develop software in a software ecosystem has led to the loss of (1) competition, (2) intellectual property, and (3) jobs in the software industry. In the study of Kilamo et al. [25], the authors discuss migrating an industrial software project from a closed source to an OSS ecosystem. An industrial case is introduced to find the supporting processes, guidelines, and best practices. To find the analogy and difference between OSS ecosystems and ecological communities, Matragkas et al. [23] perform an empirical study using GitHub ecosystems. Specifically, Matragkas et al. use cluster analysis on GHTorrent to explore the diversity as well as the structure of OSS ecosystems on GitHub. Their findings show that the percentage of core developers and active users remains stable as a project grows and most of the members of a large project are passive users.

Eckhardt et al. [26] study how meritocracy governance affects the health of a project in the Eclipse OSS ecosystem. From the management perspective, Eckhardt et al. find that the ecosystem is not always fair and merits are beneficial only in some cases. In a research by Zhang et al. [21], company participation and collaboration in the OpenStack ecosystem are studied by proposing a methodological framework. Zhang et al. characterize collaboration patterns and identify engagement strategies that companies adopt for participation. Such results may support a company in defining its own OSS strategy. Compared with existing research works, our project clarifies the term definition for the OSS ecosystem in section 3. We analyze the OSS ecosystems of influential Global 500 companies from a statistical view instead of performing case studies on a few cases, which has not been done in the previous study.

2.2 Code Change Classification

The early-stage code change classification is proposed in the late 1970s by Swanson [27] and Lientz et al. [28]. The former study gives an exploration on the dimensionality of software maintenance and proposes three dimensions to it including *corrective*, *adaptive*, and *perfective* maintenance. The latter study analyzes software maintenance categories by surveying maintenance managers. Swanson and Lientz conclude that the maintenance contains 17.4% corrective, 18.2% adaptive, 60.3% perfective, and 4.1% other efforts. A roadmap on software maintenance and evolution suggested by Bennett and Rajlich [29] shows that around 75% of the maintenance efforts are *adaptive* and *perfective*, 21% are *corrective*, and the remaining 4% are categorized as preventive works. Schach et al. [30] compare survey-based result with their empirical result based on three software products which have repeated maintenance activity.

Their findings suggest that corrective maintenance efforts are much higher than the survey-based study.

After the early exploratory stage, code change classification methods are developed to perform automatic categorization. Amor et al. [31] present a detailed classification schema for code changes based on the analysis of textual descriptions attached to each transaction in the versioning system. Similarly, Hindle et al. [32, 33] study large commits and their classification method by mining software repositories. Likewise, other authors utilize machine learning (ML) techniques to classify code changes [34, 35, 36, 37, 38]. In particular, Levin and Yehudai [34] employ 1151 labelled commits from 11 popular OSS projects to classify code changes, while Sabetta and Bezzi [35] identify security-related commits through natural language processing. [36, 38] introduce code density as a novel feature to improve the classification model. [37] add three additional features and tests with XGBoost algorithm. Compared with existing works, our project extends the dataset in work [34] with more commit messages to enrich the source of data. We also extend the keywords in [34] by exploring our dataset to find possible new keyword features to help code change classification.

2.3 Identity Disambiguation

Identity disambiguation is a technique to link data records from the same entity together. Robles and Gonzalez-Barahona [39] study the problem where developers use different identities when interacting with various tools. They propose a heuristic-based approach to find the developers who use different identities. To resolve multiple email aliases, Bird et al. [40] use hand-inspection. Steorts et al. [41] compare multiple traditional blocking techniques that link records from the same entity and also

discuss about privacy-concerned issues. Silvestri et al. [42] build a methodology for id disambiguation among social and Q&A platforms such as Stack Overflow, GitHub, and Twitter, while Amreen et al. [43] use supervised learning to detect developer identity duplications from OSS repositories. Fry et al. [44] propose a dataset containing author IDs from version control systems and a method to find all author IDs belonging to a single developer in the entire dataset. Fray et al. also share the list of all author IDs that were found to have aliases. Compared with existing works, our project mainly adopts the technique used in [43]. We experiment with different feature combinations, choose the best feature group, and build up an identity disambiguation model for our dataset.

Section 3

Definition of Terminologies

In this section, two terminologies are defined and restricted to a specific meaning in our study to eliminate the ambiguity on terminologies.

3.1 OSS Ecosystems

Zhang et al. [21] and Jansen et al. [24] define an ecosystem as a group of software users, developers, organizations, artifacts, and infrastructure interacting as a system. In our study, we refer to an OSS ecosystem of a company as a set of GitHub repositories that contain commits from its employees' email addresses. In this way, an OSS ecosystem contains most repositories within GitHub organization accounts of a company and some other collaborative repositories which do not belong to the organization accounts. An illustrative demonstration is shown in Figure 3.1. Each circle represents an organization account in GitHub with organization repositories and maintainers inside. Circles of the same colour collectively represent an OSS ecosystem of the same company. The two green circles with solid lines show that Microsoft owns two organization accounts, while the two smaller green circles with dashed lines tell us that Microsoft employees contribute to two repositories of other

companies with their working email addresses. The four green circles together define the Microsoft's OSS ecosystem. However, if an employee of Microsoft commits to a personal repository instead of an organization one, it is not counted as a part of the ecosystem as shown by the red cross in Figure 3.1.

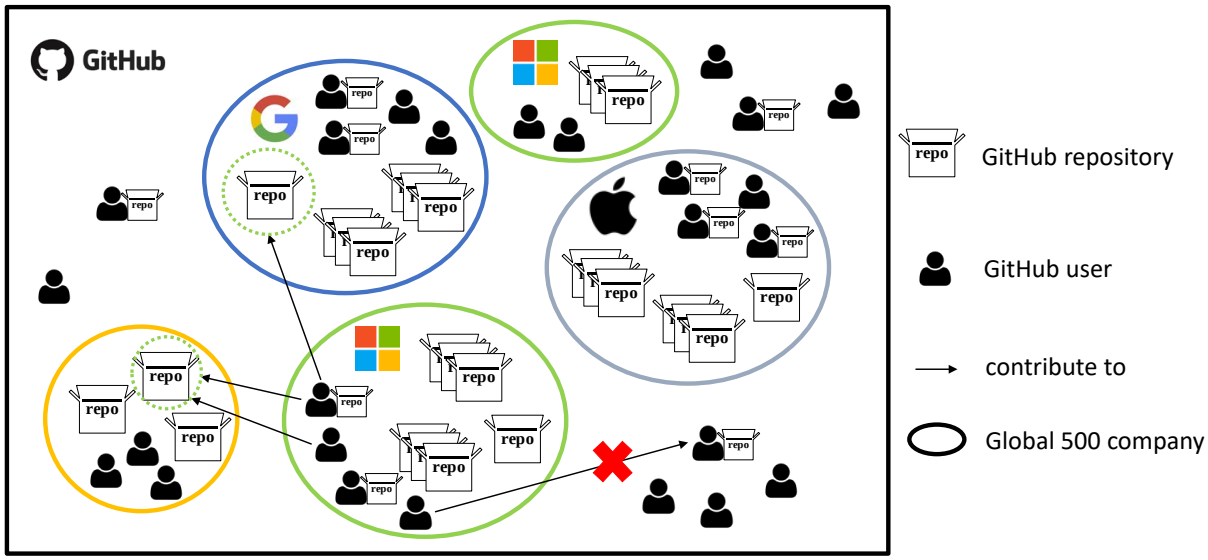


Figure 3.1: An example of OSS ecosystems on GitHub

3.2 Contributions

In our study, commits and pull requests are two aspects of the code change contribution metric. Commits are tangible code change efforts that flow into a repository to fix a bug, add a feature, refactor source code, and so on. Pull requests are attempts to give code change commits to a repository pending to be accepted by the repository's owner. When a pull request is accepted by the repository's owner, it can

be added to the repository as a code change. However, as we perform Spearman correlation analysis on the number and growth rate of commits as well as pull requests, both the number and growth rate show strong correlation based on our study finding, indicating that the two candidate metrics are repetitive. An accepted pull request is already recorded and considered as one code change commit. Therefore, we consider code change commit as the major contribution metric towards an OSS ecosystem and pull request is not taken into consideration in subsequent analysis. Other metrics, such as the number of forks, stars, and subscribers only indicate the popularity of a repository, which are not practical development efforts and are ignored.

Section 4

Experimental Setup

4.1 An Overview of Experiment Design

An overview of our empirical study is shown in Figure 4.1. As a first step, we point out all the GitHub repository names belonging to the Global 500. Then, we remove any repository not belonging to a GitHub organization account to form a list of candidate repository names. It is based on the assumption that company employees are not likely to commit to individual projects with their company email domain address [45]. With the candidate repository names, we use the REST API from GitHub to collect data.

After data collection, we find that commit and pull request are two evaluation metrics related to code changes that meet our project needs. we perform correlation analysis on these two metrics and remove a repetitive metric, pull request, from code change evaluation metrics. As soon as the code change measurement metric is finalized, we want to trace back and see who makes the code change contributions to the OSS ecosystems. We match commit messages with their contributors where in some cases the same contributor may have multiple identities, such as the email

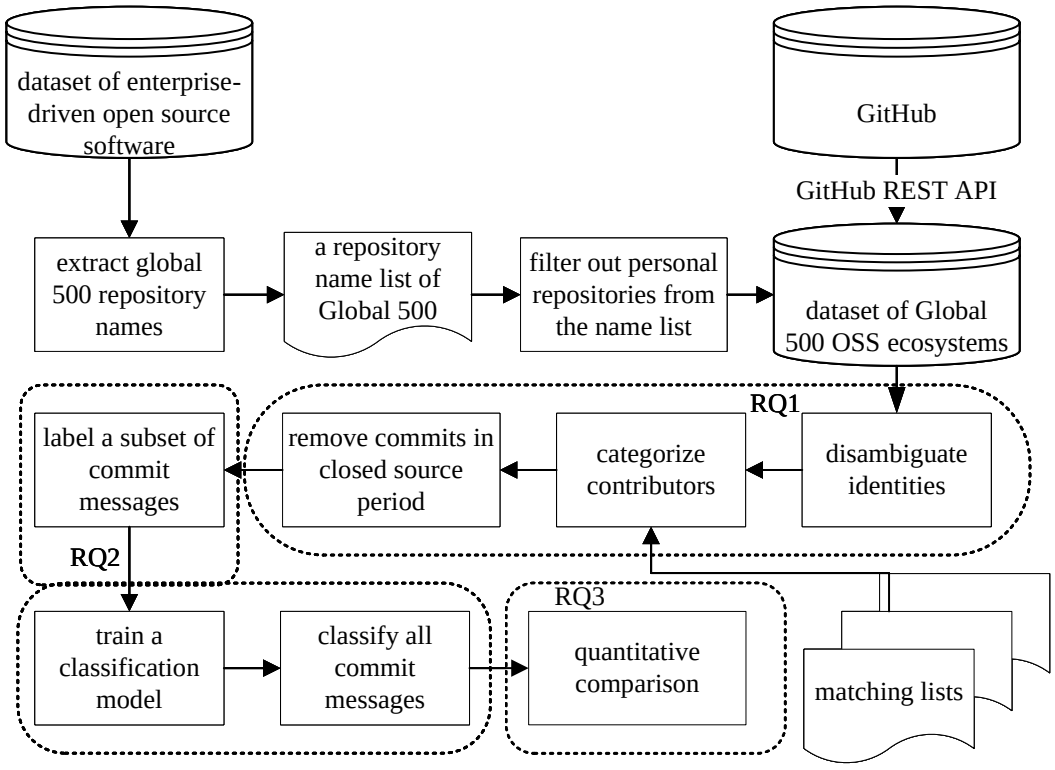


Figure 4.1: An overview of our empirical study

address, the first and last name of the commit contributor. To solve this problem, an identity disambiguation model is introduced for anti-aliasing to reduce identity errors when mapping commit messages back to their committers. Two matching lists from GitHub containing email domain information are then adopted for the matching process. Considering a repository could be initially closed-source and then transformed to open-source, we locate the closed-to-open source time in each repository and remove the commits that are contributed during the closed-source period. To get a deeper understanding of the code changes, we utilize a machine learning model to classify code changes (i.e. commits), into multiple types. We first extend an open-source

labelled dataset [34] consisting of over 1K GitHub commit messages with a subset of commit messages from our Global 500 dataset. We train and test the model based on the extended commit message training set. After having a satisfactory accuracy, we use the trained model to classify the commit messages in our dataset and analyze the distribution in each OSS ecosystem. Finally, we characterize the ecosystems by performing statistical comparisons based on various business sectors.

4.2 Data Collection

To collect our data, we execute the following three steps.

Step 1: Extracting GitHub repository names of the Global 500. First, we make use of an existing enterprise-driven open-source dataset created by Spinellis et al. [45]. Their dataset contains 17,264 enterprise GitHub repositories with some of them belonging to the Global 500. Each repository has 29 attributes, such as project URL, name, star_number, and commit_count. we match the Global 500 email domains with the repository email domains in the dataset. All enterprise-driven repositories that belong to the Global 500 are extracted. However, the dataset is outdated as it is extracted from a GHTorrent version in 2019. We strive for creating a data collection method which can always acquire the latest data information from GitHub, instead of using the outdated static snapshot provided by the existing enterprise-driven dataset.

Step 2: Collecting repository information via GitHub. To build our dataset with the latest information, we use the list of repository names extracted from the previous step and GitHub API. We remove repositories that are from GitHub personal accounts, instead of organization accounts, from the repository name list.

Table 4.1: Attributes in a code change commit and corresponding descriptions

Attribute	Description
commit_author_name	The name of the author who submits the code change commit
commit_date	The exact date and time showing when the commit is submitted
author_email	The email address that the author used for the commit
commit_message	A string that contains descriptive texts to give a brief illustration on the commit.
changed_files	Names of the files which have been changed in the commit
repository_id	The identifier of the repository to which the commit is given. In the form of author_id/repository_name
company_name	The matching company name of the commit with the enterprise-driven open-source dataset
company_email_domain	The email domain of the company which the commit belongs to

Step3: Refactoring the dataset structure for subsequent empirical studies. The newly built dataset contains 4,032,041 commits with several attributes as shown in Table 4.1. The commits come from 39 Global 500 OSS ecosystems. No GitHub organization account or commit is found for other Global 500 companies other than the 39 OSS ecosystems mentioned above. Because our initial dataset is too large, we refactor and flatten it from a JSON to a CSV file format. For each research question, we only extract a subset of necessary information, instead of taking the whole dataset for analysis, to reduce redundancy and boost analysis efficiency.

Section 5

Experimental Results

5.1 RQ1: Who contribute to the Global 500 OSS ecosystems?

Motivation: In this RQ, we aim to identify who contribute to the OSS ecosystems and explore the roles they play in the OSS ecosystem development process. Especially, it is interesting to figure out to what extent external developers participate in the ecosystem development, because one of the major goals of open sourcing is to include more code change contributions from the outside of a company, creating a vibrant open-source developing environment in the GitHub community. The more external contributors participate in an ecosystem, the more diverse an OSS ecosystem is. The results provide an outlook on what OSS ecosystems of influential companies look like, giving guidance to other companies for building up their own OSS ecosystems.

Approach: To identify the contributors of the ecosystems, identity disambiguation is used as a data preprocessing step because commit number cannot simply represent contributor number. Several code change commits could be from the same contributor. Therefore, we need to gather commits that come from the same entity into

one group, counting them as from one instead of multiple contributors. We follow the method proposed by Amreen et al. [43] and build a ML model for identity disambiguation. To train a robust model for id disambiguation, a total of 91,635 data points are randomly sampled as a training set and labelled independently by two graduate students, a PhD and a master student. The two annotators both major in software engineering with software engineering research experience. The sample size of 91,635 far exceeds the minimum sample size requirement of 385 with 95% of confidence level and 5% of margin of error. The training set is set to a large number in order to train a robust model with high accuracy. The labelled dataset is considered consistent when Cohen's Kappa value of two annotators achieves a relatively high standard of 0.98. To guarantee a balanced training set, the ratio of positive label and negative label is set approximately to 1:1.

Table 5.1: Selected features for ID disambiguation model construction

Feature	Definition
company domain match	Binary value indicating whether the code changes of two identities are given to the same company.
repository match	Binary value indicating whether the code changes of two identities are given to the same repository.
author name similarity	Jaro-Winkler similarity between two commit author names
email domain similarity	Jaro-Winkler similarity between the email domains of two identities
time zone difference	Absolute difference between time zone of two commits
first name similarity	Jaro-Winkler similarity between two first names of commit authors
last name similarity	Jaro-Winkler similarity between two last names of commit authors
username similarity	Jaro-Winkler similarity between two usernames extracted from commit author email domain

Various candidate features are tested with the training set and eight features are selected to construct the feature vector for disambiguation as shown in Table 5.1. Details are explained below. For any pair of two data points, repository id and company email domains are compared to derive two binary feature values. The *full*

name, email domain, first name, last name, and *username* of any two contributors are selected to calculate five Jaro-Winker similarities as feature values.

The Jaro similarity sim_j of two given strings s_1 and s_2 is shown as (1):

$$sim_j = \begin{cases} 0 \text{ if m} = 0 \\ \frac{1}{3}(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}) \text{ otherwise} \end{cases} \tag{1}$$

where $|s_i|$ is the length of string s_i, m is the number of matching characters, t is the number of transpositions.

The Jaro-Winkler similarity is based on Jaro similarity and the formula is shown as (2)

$$sim_w = sim_j + lp(1 - sim_j) \tag{2}$$

where l is the length of the common prefix at the start of the string, p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes.

The time zone of a commit is also considered. 24 hours within a day are divided into 24 time zones. We calculate the absolute value of the time zone difference as another feature value. It is created because commits from the same entity are expected to be submitted around a similar time period in a day. Then, we train a random forest model using the above features and dataset, which achieved an average accuracy of 99% with 10-fold cross-validation. The model is further adopted in the identity disambiguation process on the whole dataset of the Global 500 ecosystem.

After identity disambiguation, we adopt open card sort and email domain matching together to classify contributors into different groups. Since we are interested in knowing the number of external contributors from outside of a company, we initially divide contributors into two high-level categories: internal and external. Internal

Table 5.2: Groups of Contributors

Contributor Type	Group Labels	Descriptions
Internal	Own Global500	The contributor is likely to be an employee of a Global 500 company and contributes to his/her own company's OSS ecosystem.
External	Other Global500	The contributor is likely to be an employee of a Global 500 company, but he/she contributes to another Global 500 company's OSS ecosystem.
	Government	The contributor is from a government institution
	Education Institution	The contributor is from an education institution
	Individual	The contributor is not related to its occupation and is an individual
	Non-500 Corporation	The contributor is from a commercial company other than Global 500
	Non-profit Organization	The contributor is from a non-profit organization
	Network	The contributor is from a network provider
	Others	The contributor is not recognized as in any groups above

contributors submit commits to their own company's OSS ecosystem. For instance, if a developer works as an employee for Apple and submits a commit to the repository of Apple/Swift, then we consider this contributor as an internal one. Otherwise, a developer is considered an external contributor.

To detect internal contributors, we check to which OSS ecosystem a commit is given. The email domain of a company is extracted to make a comparison with the email domain of a contributor. If two domains match, it indicates that the contributor makes the code change commit to the ecosystem of his/her own company with the company's email address. The contributor is considered an internal employee of the company.

Furthermore, we divide external contributors into multiple groups based on iterative open card sort according to their email addresses. We randomly sample a subset from the dataset of the Global 500 ecosystems and examine the email domains of contributors. For the first round of card sort, we form two clusters of commercial and non-commercial groups. Then, we randomly sample each group again with 95% of confidence level and 5% of margin of error, adopting card sort to both groups to divide each group into more detailed subgroups. This process is repeated until no new

group is added to the existing group list. Several clusters are formed in the next few rounds of iteration, such as commercial company, government, education institution, non-profit organization, and network provider.

In some cases, an email domain ending with ".com" looks like from a commercial company, but is provided for personal uses, such as a personal email account. Therefore, we create a new label named individual contributors to distinguish between commercial and personal emails. A matching list [46] containing 3,782 domains of free email providers is utilized to help detect individual contributors. Also, for education institution, not all university emails end with a top domain ".edu". Instead, some end with a country code such as ".ca", ".de", ".cn", and etc. Another matching list [47] containing 9,682 university and college domains worldwide is used to help recognize email domains from education institutions and lower the detection missing rate. Moreover, we find some data pieces with incorrect email format. These external contributors which lack information and cannot be classified into the existing categories are labelled as others. Contributors are clustered into nine groups as shown in Table 5.2.

We also consider the following two situations: 1) A GitHub repository might be initially created as private with limited visibility for internal development and later become public for open-source collaboration [48]; and 2) A software project might be initially developed internally on another version control system and is migrated to GitHub repository with all previous internal commit history [49]. In the above-mentioned two situations, if a project is initially developed internally and only become open-source later, then the code change commits with their contributors during the internal development period are not included in the analysis. We assume that a

repository is in its internal development period if all commits are contributed by internal employees until there comes a code change commit from external developers. Only after then, we consider the repository turn into a real open-source status. In this way, we try to detect the close-to-open-source date by inspecting the first external commit and removing commits before that date, considering them as internal code changes given before open-source development. We filter through the nine groups and the commit history of each repository to match the first external commit. We also check GitHub user profiles and Linkedin profiles manually to guarantee that no information shows the detected contributor is an internal employee at the time of commit. For validation, we randomly sample a subset of repositories (90% confidence level and 10% margin of error), extract all commits before close-to-open-source date for each repository, and check if there is any missing external commit. If a missing external commit exists, then the deviated date between the external commit and the missing external commit is calculated. Otherwise, the deviated date is set to zero. The validation result suggests that the mean, median, and standard deviation of deviated date are 11, 0, and 39 days, which achieves relatively high accuracy. Finally, we remove the closed-source period commits from the original dataset and do statistical analysis on the result.

Results: We divide the commit number of each group by the total commit number, and divide contributor number by the total contributor number to obtain the percentage of commit and contributor in each group. The results are illustrated in Figure 5.1.

Individual and *Own Global500* groups occupy the two biggest portion of contributors in the OSS ecosystems with 39.6% and 39.5%, respectively. *Non500 Corporation*

has the third-highest proportion of contributors of 9.1%, while the remaining sectors occupy the rest 11.8% of the total contributors.

If we calculate the commit percentage without considering the identity duplication, the result shows a very different distribution. For instance, *Own Global500* occupies 70.6% of all the contributions. *Individual* group only has 13.4%, followed by *Non-profit Organization* with 5.7% and *Non500 Corporation* with 3.8%. By comparing the contributor percentage with commit percentage as shown in Figure 5.2, we find that 60.5% of contributors are from external sources other than internal employees, while the remaining are internal employees. However, when we compare the commit number between external and internal developers, it shows that internal employees contribute 70.6% of commits to the OSS ecosystems, while external developers contribute only 29.4% of commits.

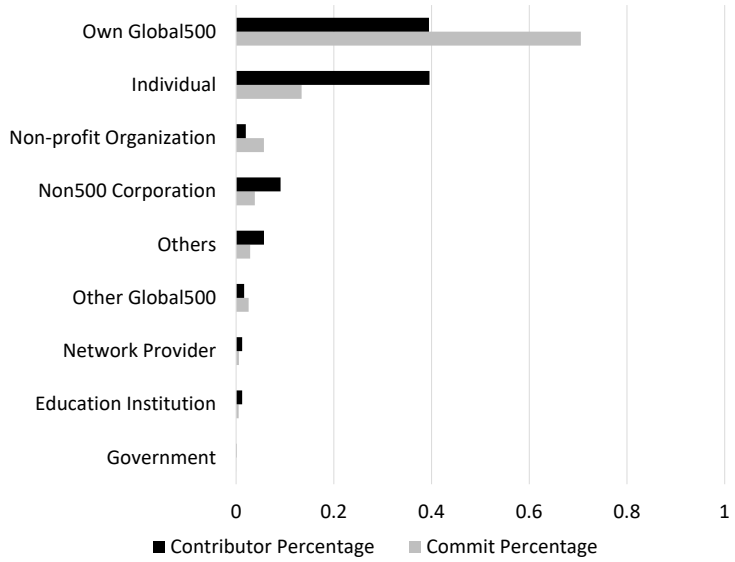


Figure 5.1: Distribution of contributor and commit based on different groups

The results depict that external developers are actively participating in

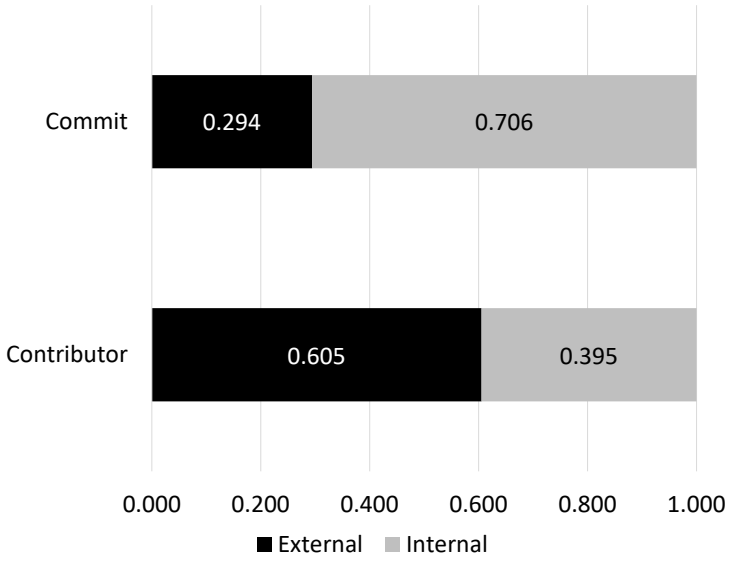


Figure 5.2: Distribution of the external and internal contributions

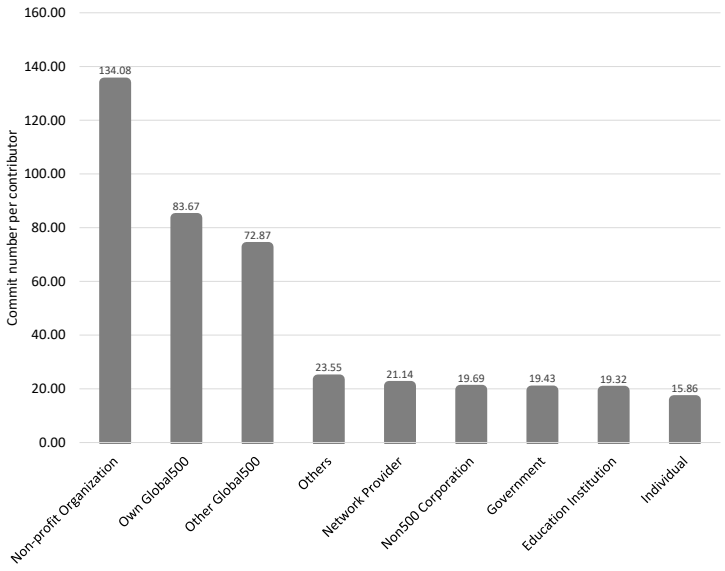


Figure 5.3: Commit number per contributor for each group

the OSS development of Global 500 ecosystems, but the code change contributions given by external developers are far fewer than internal employees with a 41.2% difference in percentage. (see Figure 5.2). This could result from various factors and does not necessarily indicate external developers make less contributions to the OSS ecosystem. One possible explanation for this phenomenon is that many code change commits by external developers come from pull requests. If a pull request is rejected by repository collaborators, then it is not counted as a code change commit to the repository. Since most repositories in Global 500 ecosystems are managed by internal employees, external pull requests that deviate from the internal development path and might be rejected in order to keep the development process consistent. Another possible reason is that external developers do not have sufficient time to contribute to the ecosystems compared to the internal employees.

We also compare the per capita code change commits, that is, the total commit number divided by the contributor number in a group, in each group as illustrated in Figure 5.3. The results show that *Non-profit Organization* group has the highest per capita commit of 134.08.

Moreover, *Own Global500* has the second and *Other Global500* has the third-highest per capita commits of 83.67 and 72.87 respectively, while the remaining sectors have similar per capita commits. **We find that developers from the *Individual* group have the lowest per capita commit number of 15.86.** We are still not sure why the *Individual* group has the highest contributor number but the lowest per capita commit number among external groups. However, more consistent contributions from the *Individual* group are needed if an OSS ecosystem wants to build up a vibrant OSS ecosystem with adequate external contributors.

External developers actively participate in the OSS *ecosystem development process, but they are counted approximately one forth of the total commit number, which is 41.2% less than internal contributors by percentage. To make an* OSS *ecosystem lively and attract more external developers, we recommend commercial companies—who build their own* OSS *ecosystem—to develop strategies to raise the code change acceptance rate of external, especially Individual developers outside the company.*

5.2 RQ2: What kinds of contributions are given to the Global 500 OSS ecosystems?

Motivation: In this RQ, we aim to identify the types of contributions given to the selected ecosystems. Commit classification is one of the major tasks. An automatic commit classification method is adopted to handle the large amount of commit messages. With the classification result, we offer information about the code change composition of the ecosystems. Practitioners can have a better understanding of the code change distributions in the ecosystems of influential companies, helping them keep track of the evolution status of their ecosystems by comparing with the Global 500 OSS ecosystems.

Approach: First, we study the types of code changes by examining the commit messages given to GitHub repositories of the Global 500. Following existing code change classification studies [27, 34], we categorize commit messages into three types: *corrective, adaptive,* and *perfective*. The classification process is divided into three subsequent steps, that is, (1) training set preparation, (2) feature vector extraction, and (3) classification model training.

To train a ML model for commit message classification, it is necessary to prepare a labelled dataset. Therefore, we utilize a publicly available dataset [34] of 1,151 labelled commit messages and extend the dataset with more commit messages from our Global 500 dataset to meet the commit classification requirement for our task. The authors of the dataset summarize 20 keywords that appear in the specific projects to build up a keyword-based classification method. But their research work only covers 11 projects. The amount of data available in their dataset is insufficient to train a ML model for the classification task on commit messages from over 2,000 projects. Therefore, we extend the dataset to train a classification model that fits more into our data distribution by adding into it more commit messages and keywords. With the confidence level of 95%, and 5% margin of error, a sample size of 385 is needed based on the sample size determination formula. Therefore, we extract 400 commit messages for labelling purposes. The labelling process is done by two annotators, a postdoctoral and a master student who both major in software engineering. The two annotators follow the software maintenance definition of Swanson [27], the classification schema of Amor et al. [31], and the labelling process of Levin and Yehudai [34]. Each code change commit is labelled with one of the three categories: (1) *corrective*, (2) *adaptive,* or (3) *perfective*. Commit messages with little textual information and unclear purposes are removed later by the annotators. After the labelling, we use Cohen's Kappa to check the labelling consistency between the two annotators who achieve a satisfactory percentage of 0.76 in labelling agreement. Then, the annotators meet and discuss about the remaining inconsistent labels to reach an agreement. All 400 commit messages are assigned with one label. After labelling and cleaning the sample, 391 labelled commits are added into the original dataset that we use to train a commit

message classification model. We adopt feature engineering to extract features of commit message keywords. We turn the commit message strings into fixed-dimension feature vectors as shown in Figure 5.4. As a first step, we tokenize and break each commit message into individual linguistic units.

Table 5.3: A running example of data preprocessing in feature extraction

1. original text:
['The bug in line 5 is fixed with No534. New features are also added']
2. tokenize the words and remove punctuations:
['The', 'bug', 'in', 'line', 'is', 'fixed', 'with', 'no', 'New', 'features', 'are', 'also', 'added']
3. transform words to lower cases:
['the', 'bug', 'in', 'line', 'is', 'fixed', 'with', 'no', 'new', 'features', 'are', 'also', 'added']
4. remove stop words:
['bug', 'line', 'fixed', 'new', 'features', 'also', 'added']
5. lemmatize words:
['bug', 'line', 'fix', 'new', 'feature', 'also', 'add']
6. retain words that match the list:
['bug', fix', 'new', add']

Then, we (1) remove all punctuation marks, (2) remove all stop-words, (3) transform all letters to lower case, (4) perform lemmatization on each word, and (5) match each remaining word with an extended keyword list. We keep words that match the keyword list and discard the ones that do not match. We calculate the occurrence frequency of all words that appear in the Global 500 dataset. The top 30 most frequent words are picked out as candidates for additional keywords. Some of the candidates that overlap with the existing keyword list are removed. From the remaining candidates, additional keywords are chosen by combination and tested by an ablation study with various keyword combinations to find the best keyword list. A running

example is demonstrated in Table 5.3. The original commit message text is "The bug in line 5 is fixed with No534. New features are also added". The text is tokenized into words and punctuations are removed. All English letters are transformed to lower cases. Stop words such as "is", "the", "with", and "no" are removed. Finally, the words are lemmatized and matched with the keyword list to retain only the matching keywords. Detailed experimental results are discussed in the following result part.

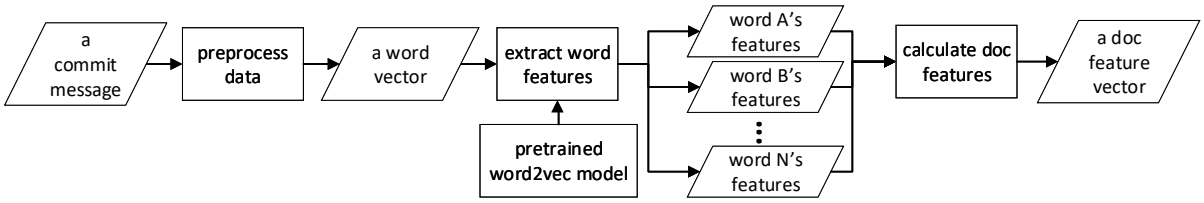


Figure 5.4: The workflow of feature extraction

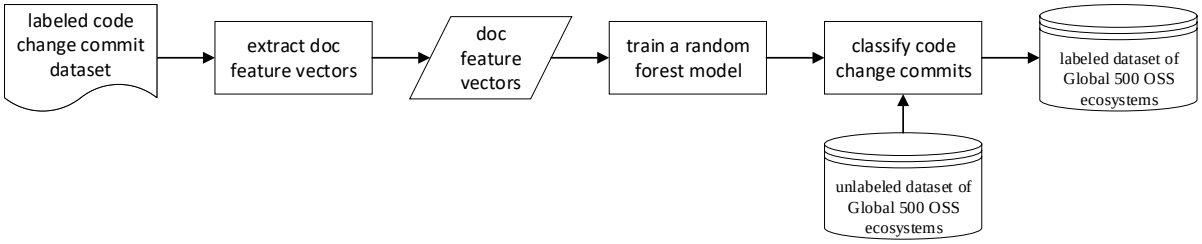


Figure 5.5: The workflow of classification model training

After preprocessing the original commit message, we acquire a vector containing several keywords. We fed each vector into a pretrained *word2vec* model provided by Google [50] to compute vector representations of words. As shown in the Figure 5.4, for a word vector with N elements, the *word2vec* model generates N feature vectors. However, the N feature vectors cannot be used directly for classification since each commit message is considered as a document, thus should have only one vector for

representation. Hence, we sum up the N word feature vectors to produce one vector which could represent the commit message. The vector is then divided by word number N for normalization.

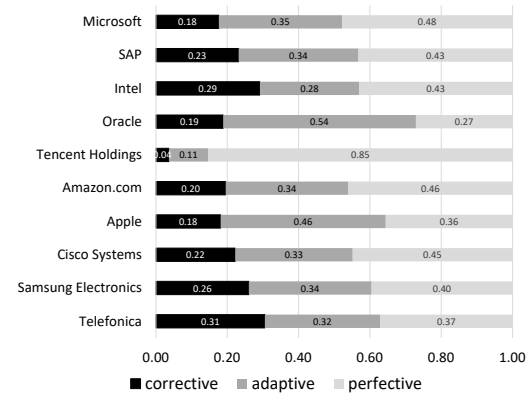
The training process of the classification model is shown in Figure 5.5. With the 1542 (1151+391) labelled training set and the feature extraction technique, each commit message string in the training set is transformed into a doc feature vector. All doc feature vectors with their labels are trained with the random forest algorithm.

Different keyword lists are used for training and testing. We select a random forest model that achieves the highest accuracy. After that, the unlabeled commit messages are fed into the trained random forest model for classification in order to generate a labelled dataset of the Global 500 OSS ecosystems.

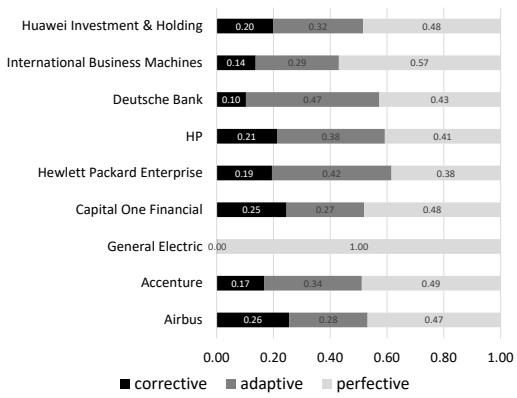
The labelled commit messages are mapped back to their OSS ecosystems. The proportions of *corrective, adaptive,* and *perfective* commits are calculated for each ecosystem. Qualitative analysis is performed on the statistical result.

Results: For commit classification, we calculate the word occurrence frequencies and analyze the top-30 most frequent words to find suitable additional keywords to add into the existing keyword list. The other less frequent words are neglected to reduce model complexity. For the existing keyword list, since "npe" is not a project-based keyword and cannot be generalized to other projects, it is removed from our keyword list. The ablation study result shows that the additional keyword combination of "merg, "updat", "md", "pull", "commit", and "code" produces the highest test performance for the model by achieving 0.675 on 10-cross validation accuracy. Thus, our extended keyword list contains 19 original keywords and 6 additional keywords.

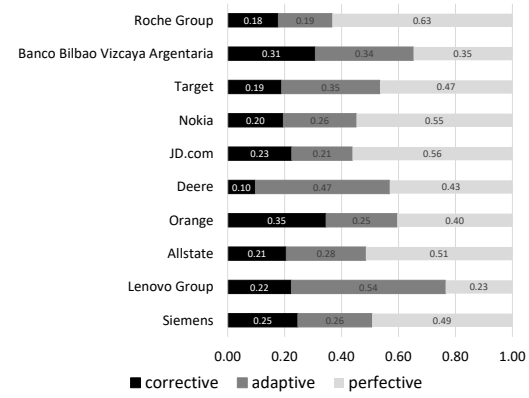
The 39 ecosystems are sorted based on their commit numbers, gathered into four



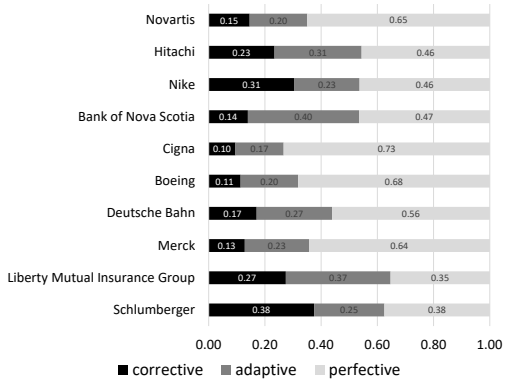
(a) Companies whose total commit numbers are above Q3



(b) Companies whose total commit numbers are between Q2-Q3



(c) Companies whose total commit numbers are between Q1-Q2



(d) Companies whose total commit numbers are below Q1

Figure 5.6: Code change percentages of perfective, corrective, and adaptive commits for 39 OSS ecosystems

groups by first, second, and third quartiles (Q1, Q2, and Q3), and calculate the percentage of three types (see Figure 5.6). **Among the 39 ecosystems, 82% of them have more perfective code change commits than any other type. The remaining 18% of ecosystems have adaptive code changes as the highest percentage. None of the ecosystems has the highest percentage of corrective code changes.** Corrective code changes remain relatively low in these

ecosystems with a stable proportion around 10%-30%. The statistical data provides a reference for companies who tend to build up their own OSS ecosystems. Corrective code changes such as bug fixing only occupies a small percentage in the Global 500 ecosystem evolution process. A large amount of perfective works are done beyond bug fixing to refine their projects. To fit into the new environment, such as cross-platform migration and hardware update, adaptive code changes also exist in these ecosystems to make the ecosystems evolve. However, a few ecosystems such as Oracle, Apple, Tencent Holdings, General Electric, Lenovo Group, and Deere demonstrate differences in the code change distribution. Oracle, Apple, Lenovo Group, and Deere have a relatively high percentage (around 50%) in adaptive code changes. **Tencent Holdings and General Electrics have a single leading code change type as perfective and very few corrective as well as adaptive code changes.** The special cases may indicate potential differences in the development mode of their ecosystems. Detailed reasons for these special cases are not delved in this study.

The results suggest that three code change composition patterns are found within the Global 500 OSS *ecosystems, which are perfective-dominant, perfective-leading, and adaptive-dominant. Perfective-dominant ecosystems have the highest percentage of perfective code changes above 33.3% and lower than 66.6%. Representatives are Microsoft, SAP, Intel, Amazon.com, etc. Perfective-leading ecosystems are the systems whose perfective code change percentage is greater than 66.6% with representatives of General Electric, Tencent Holdings, Cigna, and Boeing. The perfective code change type in these ecosystems has taken other code change types and becomes a single leading type. Adaptive-dominant ecosystems have the highest percentage of adaptive code changes but the percentage is lower than 66.6%. Representatives are Oracle, Apple, Deutsche Bank, Lenovo Group, etc. None of the ecosystems has the highest percentage of corrective code changes.*

5.3 RQ3: What are the characteristics of Global 500 OSS ecosystems?

Motivation: In this RQ, we aim to depict some characteristics of Global 500 OSS ecosystems from the statistical aspect. We propose a list of questions as follows.

1) What are the popular project topics in the ecosystems?

2) How old are the projects in the ecosystems?

3) How frequent are the projects in the ecosystems contributed by developers?

By answering the questions, we manage to characterize the OSS ecosystems. We hope to demonstrate these characteristics to practitioners in different business sectors, helping them understand to what extend the OSS environment is adopted in their business area.

Approach: First, we characterize the OSS ecosystems by dividing all ecosystem

projects based on business sectors with the information provided in [45]. 10 business sectors are found in the Global 500 dataset initially, but some of them have too few data points for the statistical analysis and such sectors are dropped. Finally, we obtain 6 business sectors for comparison, including Financials, Health Care, Industrials, Retailing, Technology, and Telecommunications.

We answer question 1 by inspecting the types of repositories in the dataset manually. We follow an open labelling process to assign multiple labels to each repository. The labels have no priority among each other. Sometimes, a GitHub repository can be assigned multiple tags by its owner, showing to which areas the repository belongs. If there are tags that exist in a GitHub repository, then we pick a few tags that are representative of the repository content. Otherwise, the description and README file of the repository are examined to create tags corresponding to the repository. When the tagging process is finished, we calculate the occurring frequency of all tags. Tags of similar meaning are merged to reduce duplication of the result. We analyze the popular topics on two different levels. First, we analyze the popular topics based on all ecosystems projects, which gives us a high-level overview of the popular topics. Considering that the Global 500 companies are of different business types, the popular topics could vary among different business types. We also analyze the popular topics in each business sector.

For question 2, we analyze the age distribution of repositories using bean plots for each business sector. From each bean plot curve, we count the number of peaks shown in the curve. Each peak indicates an open-source wave where many repositories are open-sourced intensively in a period of time. With the peak number, we can know exactly the number of open-source waves each sector experiences.

For question 3, we try to make a fair comparison on the contributions of different OSS ecosystems— since some of them have longer ages than others— we divide the number of commits of each repository by the age of repository to acquire the commit growth rate as shown in Equations 1.

$$r_{c_i} = \frac{nc_{R_i}}{d_{R_i}} \tag{1}$$

The symbol r_c represents the growth rate of commits. R_i is a GitHub repository. nc_{R_i} represents the number of commits of R_i. d_{R_i} is the age of repository R_i in days.
Results: For repository categorization, the open labelling process finds 102 tags in total. Considering there are some semantically similar tags, we manually verify and merge the semantically similar tags, resulting in 95 different categories. After sorting based on the tag occurrence frequency, the top 10 most popular project topics are shown in Table 5.4.

Table 5.4: Top 10 most frequently occurring tags

Framework	516	Cloud Service	264
Programming Language	173	Example	163
Documentation	146	App	121
Container	103	Code Editor	93
Data Science	92	DevOps	88

The results suggest that popular projects within Global 500 ecosystems cover a broad range of interests. **The Global 500 companies actively contribute to the software development stages such as coding, testing, deployment, and maintenance within the software development life cycle (SDLC).** *Programming language* and *code editor* are related to computer language and integrated development environment for software coding stage. *DevOps* and *container* are popular

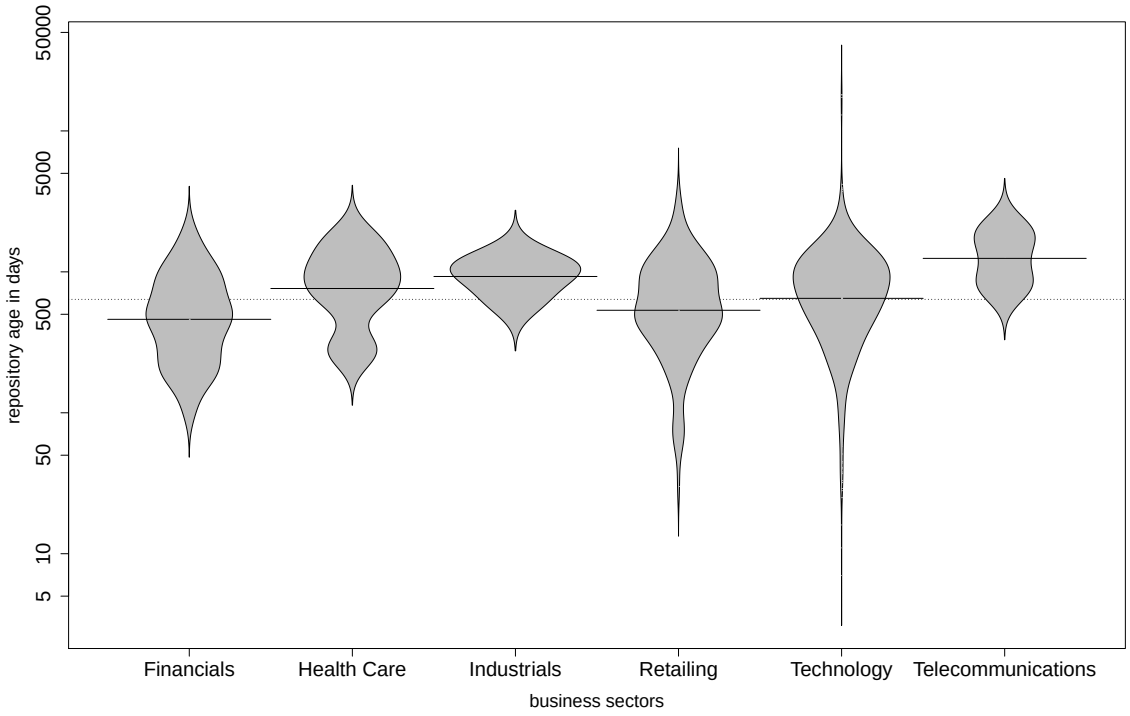


Figure 5.7: Bean plots for the age of repository

techniques related to software testing, deployment, and maintenance. *Cloud services* is related to providing infrastructures for collaborative software development.

Apart from a high-level view of the popular trend within all ecosystems, we also manage to see the popular topics for the ecosystems based on each business sector. We exclude tags that are commonly seen in most project development, such as *framework (also include library, sdk, api, interface)*, *example*, and *documentation*, to obtain more distinctive tags for different business sectors. **For *technology* and *retailing* sectors, *cloud service* still remains the top heated developing trend. *Telecommunications*, *financials*, *health care* sectors focus on *data*,**

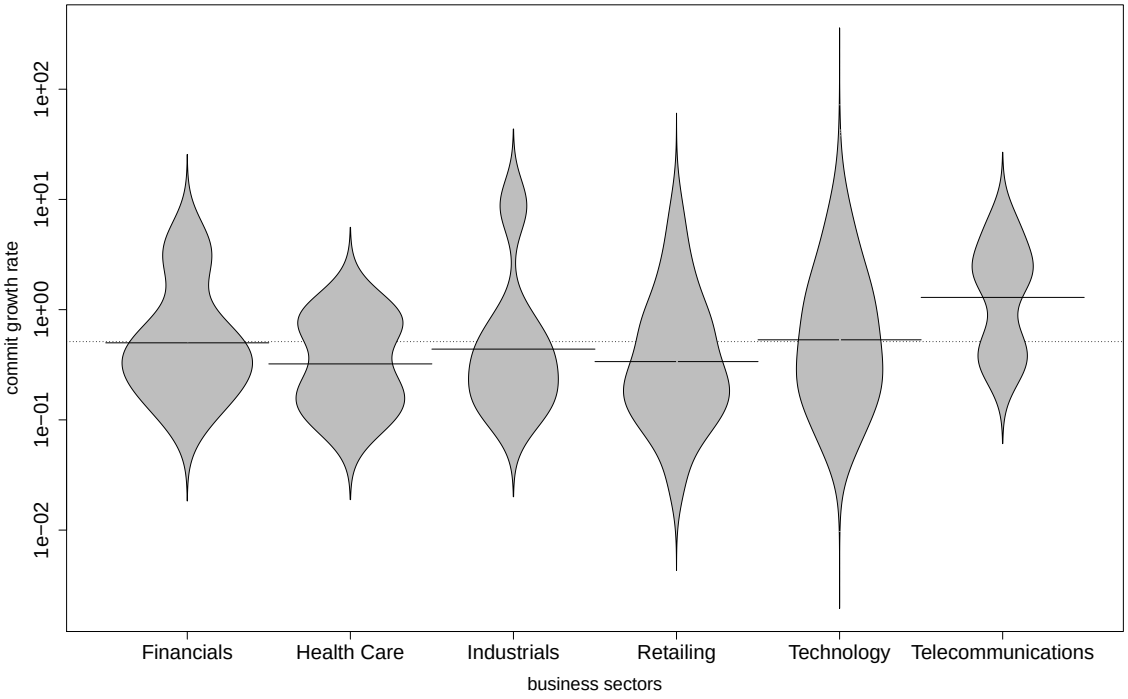


Figure 5.8: Bean plots of commit growth rate

***security*, and *bioinformatics* related projects respectively. *Industrial* sector has a unique focus on *robot* and *IoT*.** The result suggests that different business sectors focus on their unique topics. The popular topics are not universal for different business sectors.

The bean plots for the age of repository are shown in Figure 5.7. For the age of repository, we measure the central tendency of the data distribution with the mean value instead of the median. It is because we have a few data points with significantly long age, but the age of most data points are short. The median value is likely to be biased and neglect the impact of the data points with long age as it only takes the value of the middle data point. We compare the mean values of different sectors and

find that *telecommunications* sector has the longest age of 1348 days (3.7 years) while *financials* has the shortest of 585 days (1.6 years). *Industrial*, *health care*, *technology*, and *retailing* have 956, 908, 867, and 711 days respectively. **No matter for which business sector, there is an open-source wave around 1.6 to 3.7 years based on the mean value of repository age. After the intensive open-source wave, the creation of new open-source repositories appears to be less and less.** We calculate the most densely distributed part, that is, the point of repository age with the highest probability density value, of each bean plot. The point indicates the peak of the largest open-source wave. The peaks of the largest open-source waves are 357, 800, 1087, 430, 495, and 970 days for *financials*, *health care*, *industrial*, *retailing*, *technology*, and *telecommunications* respectively. With the peak points, we could locate the time when the largest open-source wave happens. The result of the bean plots shows that some sectors such as *financials*, *industrial*, *retailing*, *technology*, and *telecommunications* have only one open-source wave. *Health care* has a big open-source wave at first and a second small one afterwards. None of them are detected to have a second open-source wave greater than the first time. It could indicate that as an ecosystem experiences an open-source wave, it gradually achieves a relatively mature state with only a few repository additions needed afterwards.

The bean plots of the growth rate of commits are shown in Figure 5.8. *Telecommunications* has the highest r_c of 2.10 commits per day on average while *health care* has the lowest of 0.46 commits per day. ***Technology* sector has a few repositories with significantly high r_c, indicating these repositories have been developed actively.**

The OSS *ecosystems of Global 500 have the following characteristics.* **1)** *Projects related to software coding, testing, deployment, and maintenance are popular trends within the ecosystems.* **2)** *Ecosystems in different business sectors have different application focuses. Technology and retailing sectors focus on cloud services the most. Telecommunications, financials, and health care put more efforts in data, security, and bioinformatics relavant topics respectively. Industrial majors in robot and IoT development.* **3)** *Ecosystems in all business sectors have only one peak wave of creating open-source repositories. A few have a small second wave but none exceeds the first one by scale and length.* **4)** *Ecosystems in technology sector have a few repositories with significantly high commit growth rate, showing that the ecosystems contain some influential projects and attract a large amount of code change commits.*

Section 6

Threats to Validity

6.1 Internal Validity

The internal threats to validity come from three parts: (1) our label consistency for code change commits, (2) our keywords selection, and (3) classification model accuracy. The 391 commit messages used to extend an existing dataset are labelled by two graduate student annotators. As the annotators could have different levels of software development experience and are not the same as the annotators in the original data set, it could increase the potential risks of inconsistency with the labels in the original dataset. Our keywords selection is only limited to the top 30 most frequently appeared keywords. The keywords other than the top 30 are not considered as we assume that the less frequent a word appears in the dataset, the less it affects the classification result. With empirical observation, the top 30 keywords are sufficient to characterize and classify the texts in our dataset. However, the assumption could miss a few keywords that might have a significant impact on the classification task. The random forest model achieves relatively good performance for the classification task on the test set. But more text classification models could be experimented to

further improve the classification accuracy. The above-mentioned problems could be addressed with the following methods: (1) including more annotators to ensure a more objective labelling process, (2) trying more keyword combinations including the words behind top 30, and (3) improving the classification accuracy by choosing better machine learning models and adopting more feature dimensions.

6.2 External Validity

Threats to external validity are concerned with the generalizability of our approach as well as the findings. In our study, we use more than two thousand GitHub repositories to conclude our findings. The conclusions are drawn with relatively comprehensive statistical analysis on GitHub community. But the methodology might not be replicable to other open-source communities since not all the attributes in our study are accessible in other OSS ecosystems.

Section 7

Conclusion

7.1 Conclusion

In this research, we conduct an empirical study on the OSS ecosystems of the Global 500 companies. The analysis shows that 21.0% more external developers participate in OSS ecosystems development than company employees do. But external developers have less commit contributions than internal developers of 41.2% in percentage. In particular, the *Individual* group has the most contributors but the least per capita commits among all external contributor groups. We suggest commercial companies develop strategies to raise the code change acceptance rate of external developers to maintain a lively OSS ecosystem. By analyzing code change commit compositions, three patterns of perfective-dominant, perfective-leading, and adaptive-dominant are detected with their representative ecosystems. The result provides practitioners a view on potential code change composition patterns of mature OSS ecosystems. Moreover, we conclude four characteristics of the ecosystems as 1) developing actively on topics relevant to software development life cycle; 2) having

different application focuses for ecosystems in different business sectors; 3) experiencing only one major open-source wave to become stable, and 4) having repositories in *technology* sector with significantly high commit growth rate. Practitioners working in different business areas could make use of our corresponding analysis divided by business sectors to develop their own OSS ecosystems.

7.2 Future Work

In the future, we will perform some case studies and interviews on the companies which are found been heavily contributed to or with a unique code change composition. We would like to verify our data analysis result and derive more concrete guidelines to help small commercial companies build up their OSS ecosystems.

Bibliography

[1] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, "Understanding free/open source software development processes," 2006.

[2] V. Midha and P. Palvia, "Factors affecting the success of open source software," *Journal of Systems and Software*, vol. 85, no. 4, pp. 895–905, 2012.

[3] A. Fuggetta, "Open source software—-an evaluation," *Journal of Systems and software*, vol. 66, no. 1, pp. 77–90, 2003.

[4] M.-W. Wu and Y.-D. Lin, "Open source software development: an overview," *Computer*, vol. 34, no. 6, pp. 33–38, 2001.

[5] P. V. Singh, "The small-world effect: The influence of macro-level properties of developer collaboration networks on open-source project success," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 2, pp. 1–27, 2010.

[6] S. Androutsellis-Theotokis, D. Spinellis, M. Kechagia, G. Gousios *et al.*, "Open source software: A survey from 10,000 feet," *Foundations and Trends in Technology, Information and Operations Management*, vol. 4, no. 3-4, pp. 187–347, 2011.

[7] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *IEEE transactions on software engineering*, vol. 30, no. 4, pp. 246–256, 2004.

[8] A. Bonaccorsi and C. Rossi, "Why open source software can succeed," *Research policy*, vol. 32, no. 7, pp. 1243–1258, 2003.

[9] G. Von Krogh and E. Von Hippel, "The promise of research on open source software," *Management science*, vol. 52, no. 7, pp. 975–983, 2006.

[10] M. Aberdour, "Achieving quality in open-source software," *IEEE software*, vol. 24, no. 1, pp. 58–64, 2007.

[11] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas, and I. Stamelos, "Evaluating the quality of open source software," *Electronic Notes in Theoretical Computer Science*, vol. 233, pp. 5–28, 2009.

[12] K. Ven, J. Verelst, and H. Mannaert, "Should you adopt open source software?" *IEEE software*, vol. 25, no. 3, pp. 54–59, 2008.

[13] J. West and S. Gallagher, "Challenges of open innovation: the paradox of firm investment in open-source software," *R&d Management*, vol. 36, no. 3, pp. 319–331, 2006.

[14] M. Andersen-Gott, G. Ghinea, and B. Bygstad, "Why do commercial companies contribute to open source software?" *International journal of information management*, vol. 32, no. 2, pp. 106–117, 2012.

[15] T. O'Reilly, "Lessons from open-source software development," *Communications of the ACM*, vol. 42, no. 4, pp. 32–37, 1999.

[16] Google, "Google open source," 2020. [Online]. Available: https://opensource.google/

[17] Microsoft, "Microsoft open source," 2021. [Online]. Available: https://opensource.microsoft.com/

[18] Facebook, "Facebook open source," 2021. [Online]. Available: https://opensource.fb.com/

[19] P. S. Kochhar, E. Kalliamvakou, N. Nagappan, T. Zimmermann, and C. Bird, "Moving from closed to open source: observations from six transitioned projects to github," *IEEE Transactions on Software Engineering*, 2019.

[20] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 2017, pp. 186–196.

[21] Y. Zhang, M. Zhou, K.-J. Stol, J. Wu, and Z. Jin, "How do companies collaborate in open source ecosystems? an empirical study of openstack," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1196–1208.

[22] J. Linåker, P. Rempel, B. Regnell, and P. Mäder, "How firms adapt and interact in open source ecosystems: Analyzing stakeholder influence and collaboration patterns," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2016, pp. 63–81.

[23] N. Matragkas, J. R. Williams, D. S. Kolovos, and R. F. Paige, "Analysing the

'biodiversity' of open source ecosystems: the github case," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 356–359.

[24] S. Jansen, A. Finkelstein, and S. Brinkkemper, "A sense of community: A research agenda for software ecosystems," in *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 2009, pp. 187–190.

[25] T. Kilamo, I. Hammouda, T. Mikkonen, and T. Aaltonen, "From proprietary to open source—growing an open source ecosystem," *Journal of Systems and Software*, vol. 85, no. 7, pp. 1467–1478, 2012.

[26] E. Eckhardt, E. Kaats, S. Jansen, and C. Alves, "The merits of a meritocracy in open source software ecosystems," in *Proceedings of the 2014 European Conference on Software Architecture Workshops*, 2014, pp. 1–6.

[27] E. B. Swanson, "The dimensions of maintenance," in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 492–497.

[28] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of application software maintenance," *Communications of the ACM*, vol. 21, no. 6, pp. 466–471, 1978.

[29] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, 2000, pp. 73–87.

[30] S. R. Schach, B. Jin, L. Yu, G. Z. Heller, and J. Offutt, "Determining the distribution of maintenance categories: Survey versus measurement," *Empirical Software Engineering*, vol. 8, no. 4, pp. 351–365, 2003.

[31] J. J. Amor, G. Robles, J. M. Gonzalez-Barahona, and A. Navarro, "Discriminating development activities in versioning systems: A case study," in *Proceedings PROMISE*, vol. 2006. Citeseer, 2006, p. 2nd.

[32] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us? a taxonomical study of large commits," in *Proceedings of the 2008 international working conference on Mining software repositories*, 2008, pp. 99–108.

[33] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classication of large changes into maintenance categories," in *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 2009, pp. 30–39.

[34] S. Levin and A. Yehudai, "Boosting automatic commit classification into maintenance activities by utilizing source code changes," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 97–106.

[35] A. Sabetta and M. Bezzi, "A practical approach to the automatic classification of security-relevant commits," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2018, pp. 579–582.

[36] S. Hönel, M. Ericsson, W. Löwe, and A. Wingkvist, "Importance and aptitude of source code density for commit classification into maintenance activities," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 109–120.

[37] R. V. Mariano, G. E. dos Santos, M. V. de Almeida, and W. C. Brandão, "Feature changes in source code for commit classification into maintenance activities," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 515–518.

[38] S. Hönel, M. Ericsson, W. Löwe, and A. Wingkvist, "Using source code density to improve the accuracy of automatic commit classification into maintenance activities," *Journal of Systems and Software*, vol. 168, p. 110673, 2020.

[39] G. Robles and J. M. Gonzalez-Barahona, "Developer identification methods for integrated data from various sources," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[40] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*, 2006, pp. 137–143.

[41] R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg, "A comparison of blocking methods for record linkage," in *International conference on privacy in statistical databases*. Springer, 2014, pp. 253–268.

[42] G. Silvestri, J. Yang, A. Bozzon, and A. Tagarelli, "Linking accounts across social networks: the case of stackoverflow, github and twitter." in *KDWeb*, 2015, pp. 41–52.

[43] S. Amreen, A. Mockus, R. Zaretzki, C. Bogart, and Y. Zhang, "Alfaa: Active learning fingerprint based anti-aliasing for correcting developer identity errors

in version control systems," *Empirical Software Engineering*, vol. 25, no. 2, pp. 1136–1167, 2020.

[44] T. Fry, T. Dey, A. Karnauch, and A. Mockus, "A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits," in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 518–522.

[45] D. Spinellis, Z. Kotti, K. Kravvaritis, G. Theodorou, and P. Louridas, "A dataset of enterprise-driven open source software," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 533–537.

[46] T. B. Jones, "tbrianjones/free_email_provider_domains.txt," 2021. [Online]. Available: https://gist.github.com/tbrianjones/5992856

[47] Hipo, "University domains and names data list & api," 2021. [Online]. Available: https://github.com/Hipo/university-domains-list

[48] G. Pinto, I. Steinmacher, L. F. Dias, and M. Gerosa, "On the challenges of open-sourcing proprietary software projects," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3221–3247, 2018.

[49] I. GitHub, "Setting repository visibility," 2021. [Online]. Available: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/managing-repository-settings/setting-repository-visibility

[50] Google, "word2vec," 2021. [Online]. Available: https://code.google.com/archive/p/word2vec/

[51] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean ghtorrent: Github data on demand," in *Proceedings of the 11th working conference on mining software repositories*, 2014, pp. 384–387.

[52] GitHub, "Code search," 2021. [Online]. Available: https://github.com/search