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

Anonymous Author(s)*

ABSTRACT

The collaborative and sharing characteristics of Open-Source Software (OSS) have attracted commercial companies to make their projects open-source and attract external contributions from developers outside the company. As the number of open-source projects and contributors grows in a company, an open-source ecosystem gradually comes into being. Due to the high financial and time cost of building up an OSS ecosystem from the scratch by trial-and-error, there is a demand in characterizing successful ecosystems and summarizing a few guidelines for other companies who want to develop their own OSS environment to avoid invalid effort and reduce cost. In this paper, we focus on Fortune Global 500, the 500 most influential commercial companies worldwide, and measure the contributions given to their OSS ecosystems. We analyze over 4 million commit messages from more than 2,000 GitHub repositories. Our study shows that 40 companies of the Global 500 build up their OSS ecosystems on GitHub to develop open-source software. Six OSS ecosystems are heavily-contributed with high commit growth rates. Moreover, we offer a statistical view on the commit composition of these OSS ecosystems, giving practitioners some ideas on which direction a newly built OSS ecosystem should evolve to. We also compare the number of contributor and commit between external and internal developers. Although there are more external than internal developers participate in the OSS development process, external developers have much lower commit number in total. For improvement, we appeal to the companies to raise external commit acceptance rate to maintain a lively OSS ecosystem. The result of our study can potentially assist commercial companies build up their OSS ecosystems efficiently by characterizing existing OSS ecosystems in Global 500 through big data.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; **Software libraries and repositories**; **Empirical software validation**.

KEYWORDS

Open-Source Software, Fortune Global 500, Software Repository Mining, GitHub

ICSE 2022, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Anonymous Author(s). 2022. An Empirical Study on Open Source Ecosystems of Fortune Global 500 Companies. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Open-Source Software (OSS) is a paradigm developed out of the proprietary software model [13, 30, 35, 46]. It brings together software developers from all around the world regardless of their geographic, language, time zone, and firm boundaries [5, 38]. Such fact increases the working flexibility and efficiency [32] 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 [1, 8, 39, 44].

As the open-source paradigm gradually been adopted by more and more developers, commercial companies start to build business models fitting into the open innovation environment [43]. Many companies set up their own OSS ecosystems to attract project contributions through external practitioners [4, 31, 45]. Tech-giants like Google [14], Microsoft [29], and Facebook [11] have already built such OSS ecosystems and open sourced some of their products such as Google's Android, Facebook's React, and Microsoft's ASP.NET framework [23].

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 [9]. Although tech-giants could affort 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 guideline to help companies build up ecosystems more cost-saving and avoid invalid efforts.

We search through studies related to OSS ecosystems to see if there is any inspiration on filling the gap. Zhang et al. [47] 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. [26] perform a case study on Apache Hadoop ecosystem to explore changing stakeholder influence and collaboration patterns. Their findings show that both influence shifting and collaborations exist between rivaling and non-competing firms. Matragkas et al. [28] 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. None of them have analyzed GitHub ecosystems of Global 500, where many tech-giant companies reside. There is a gap in characterizing influential enterprise-based OSS ecosystems with big data and summarizing useful guidelines that help build up an OSS ecosystem efficiently.

To fill the gap, our study proposes to investigate the Global 500's OSS ecosystems with their code change contributions. In our empirical study, we draw statistical conclusions by analyzing over four million GitHub commit messages from more than two thousand repositories, showing practitioners statistical facts on the OSS ecosystems of influential companies. We also strive for providing some useful guidelines in building up an attractive OSS ecosystem. Therefore, we set the following research questions:

- **RQ1: How many contributions are given to the OSS ecosystems of Global 500?**—The solution to this RQ shows that 40 OSS ecosystems have been developed in the wave of OSS movement. Ecosystems that are heavily contributed are Microsoft, Intel, SAP, Oracle, Amazon, and Tencent Holdings. The RQ points out that if other companies want to build their own OSS ecosystems, the six heavily contributed ecosystems could be good samples for reference.
- **RQ2: What kinds of repositories and contributions reside in the OSS ecosystems of Global 500?**—The result shows that *container, DevOps, cloud service, programming language, artificial intelligence, data science,* and *IoT* are popular repository topics within the ecosystems. It also shows that 90% of the ecosystems have more perfective and 10% have more adaptive commits than any other types. The results of this RQ give practitioners some ideas of what code change composition of a mature OSS ecosystem looks like. Moreover, it helps newly-built OSS ecosystems clarify their evolution directions.
- **RQ3: Who gives contributions to the OSS ecosystems of Global 500?**—We study the identities of contributors to understand who are the major driving forces of the Global 500 OSS ecosystems. The result shows that 41.6% of contributors are company employees who contribute to their own company's OSS ecosystem. 58.4% of contributors are external developers other than company employees. External developers only contribute 26.6% of the total commit number. 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.

Paper Organization: The rest of the paper is organized as follows. In Section 2, we give definitions of two terminologies. In Section 3, we describe the overall workflow and data collection process. In Section 4, we discuss our results and in Section 5 we present the threats to validity of our study. In Section 6, we summarize prior studies related to OSS ecosystem, code change categorization, and identity disambiguation. Finally, we conclude and present our future works in Section 7.

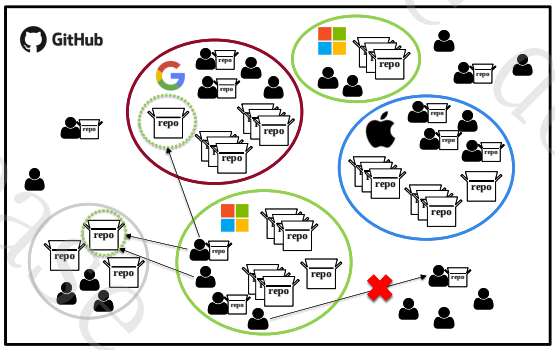

Figure 1: An example of OSS ecosystems on GitHub

2 TERM DEFINITIONS

OSS Ecosystems: Zhang et al. [47] and Jansen et al. [20] define an ecosystem as a group of software users, developers, organizations, artifacts, and infrastructure interacting as a system. In our study, we refer to a company's OSS ecosystem 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 1. Each circle represents an organization account in GitHub with organization repositories and maintainers inside. Circles of the same color 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 1.

Contributions: In our study, commits and pull requests are two aspects of code change contribution measurement. 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 attemptions to give code change commits to a repository pending to be accepted by the repository owner. When a pull request is accepted by the repository 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. Moreover, an accepted pull request is already recorded and considered as one code change commit. So 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 number of forks, stars, and subscribers only indicate the popularity of a repository, which are not practical development efforts and should be ignored.

3 METHODOLOGY

3.1 Overall Workflow

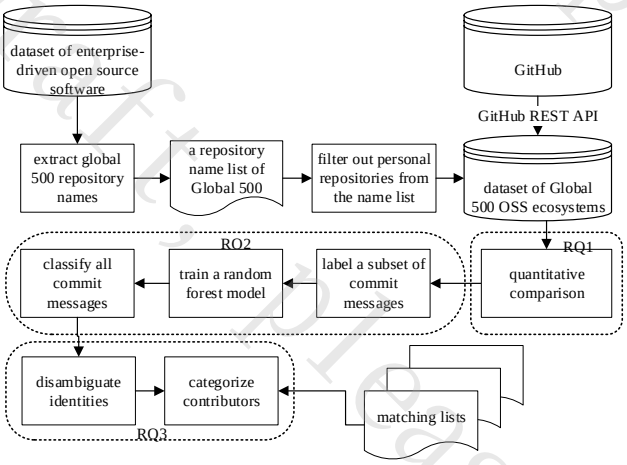

Figure 2: An overview of our empirical study

An overview of our empirical study is shown as Figure 2. As a first step, we point out all the GitHub repository names belonging to the Global 500. Then, we remove any repositorie not belonging to a GitHub organization account to form a list of candidate repository names. It is based on the instinct that company employees are not likely to commit to individual projects with their company email domain address [40]. With the candidate repository names, we use GitHub's REST API to collect our data.

After data collection, we perform correlation analysis to remove a repetitive metric for code change measurement. We also compare code change growth rate among the OSS ecosystems to see which repositories attract code changes the most. To get a deeper understanding of what the code changes are, we utilize a machine learning model to classify code changes (i.e. commits), into multiple types. We first train and test the model to an open-source labelled dataset consisting of over 1K GitHub commit messages. After having a satisfactory accuracy, we use the trained model to classify our dataset's commit messages and analyze their distribution in each OSS ecosystem. As soon as the commit types with their distribution are clarified, we want to trace back and see who make the code change contributions. We match commit messages with their contributors where in some case the same contributor may have multiple identities, such as committer's email address, committer's first and last name. 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 used to process the matching process.

3.2 Data Collection

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

Step 1: Extract GitHub repository names of the Global 500. First, we make use of an existing enterprise-driven open-source dataset created by Spinellis et al. [40]. Their dataset contains 17,264 enterprise GitHub repositories with some of them belonging to the Global 500. Each repository has 29 attributes including project URL, name, star_number, commit_count, and etc. 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 tend to create 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: Collect repository information via GitHub's API. To build our dataset that has the latest information, we use the list of repository names extract from the previous step and GitHub's API. We remove repositories that are from GitHub personal account, instead of organization account, from the repository name list.

Step3: Refactor dataset structure for subsequent empirical study. The newly built dataset contains 4,032,041 commits with several attributes as shown in Table 1. Because our initial dataset was too large (*i.e.,* 164 GiB), we refactored and flattened it from a JSON to a CSV file format. Only necessary information is selected to create the refactored dataset to reduce redundancy and boost analysis efficiency.

4 ANALYSIS OF RESULTS

RQ1: How many contributions are given to the OSS ecosystems of Global 500?

Motivation: As an initial exploration, it is unclear how many companies have built up their own OSS ecosystems. Also, there is no quantitative measurement to evaluate contributions given to the Global 500 OSS ecosystems. In RQ1, we choose two metrics to measure code change contributions. We also analyze their correlationship to remove a repetitive metric. Quantitative analysis is carried out with the chosen metric.

Approach: We calculate the number of Global 500 companies exist in our dataset, determining how many of the companies have built up their OSS ecosystems, that is, having at lease one repository on GitHub under their email domain. To measure the contributions to an OSS ecosystem quantitatively, we analyze the candidate metrics of commit and pull request contributed to its repositories.

To have a fair comparison among the contributions of different OSS ecosystems—since some of them have longer lifetime than others—we divide the number of commits and pull requests of each repository by the repository's lifetime length to acquire the (1) commit growth rate and (2) pull request growth rate as shown in Equations 1 and 2.

$$r_c = \sum_{R_i \in S} \frac{nc_{R_i}}{d_{R_i}} \quad (1)$$

$$r_p = \sum_{R_i \in S} \frac{np_{R_i}}{d_{R_i}} \quad (2)$$

The symbols r_c and r_p represent the growth rate of commits and pull requests respectively. $S = \{R_1, R_2, ..., R_n | n \in N*\}$ is a set of repositories that belong to an OSS ecosystem and R_i is a GitHub repository. nc_{R_i} and np_{R_i} represent the number of commits and pull

Table 1: Dimensions of attributes in a code change commit and corresponding descriptions

Attribute Dimension	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 containing descriptive texts that gives 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

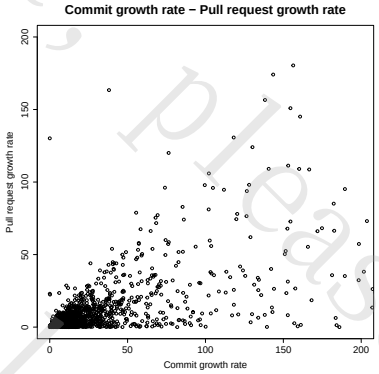

Figure 3: Scatter plot of commit and pull request growth rate

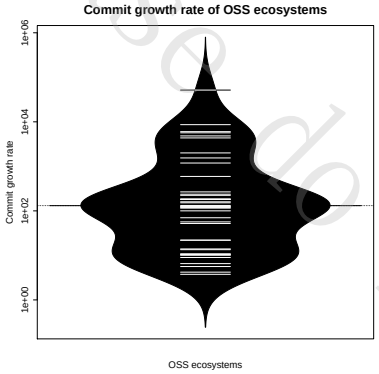

Figure 4: Bean plot of commit growth rate

requests of R_i. d_{R_i} is the lifetime of repository R_i in months. We choose monthly instead of daily or yearly growth rate because: (1) some repositories are created recently and have a lifetime less than a year, (2) code changes might not be contributed to a repository every day. The number of code changes could fluctuate between week days and weekends. In this situation, daily growth rate might be too narrow for measurement. Adopting monthly commit growth rate is a better choice.

We consider an OSS ecosystem's commit growth rate the sum of its repositories commit growth, while we calculate the growth rate of pull requests in a similar way. To avoid choosing repetitive metrics, a Spearman correlation analysis is performed on r_c and r_p. After eradicating the repetitive metric, a bean plot is drawn to show the contributions given to each OSS ecosystem.

Results: With statistical analysis, we find 2,201 repositories that come from 40 of the Global 500 companies. We consider these 40 companies have built up their OSS ecosystems on GitHub while the others have not. Further analysis is performed on the 40 OSS ecosystems with their repositories. The correlation coefficient of commit growth rate r_c and pull request growth rate r_p is 0.72, and the p-value approximately equals to zero in two-tail distribution (less than 0.05). The statistical values demonstrate a strong correlation between r_c and r_p. A similar strong correlation (correlation coefficient of 0.75 and p-value of 0) is also found between the number of commits and pull request. The strong correlations indicate that commits and pull requests share a close relationship in code change number and growth rate. The scatter plot of r_c and r_p is also shown in Figure 3. The scatter plot has most points at the left-down corner, which means that most repositories have relatively low commit and pull request growth rate. Since the two candidate metrics share a strong correlation and pull requests accepted by repository owners are included in code change commits. We abandon pull request and mainly focus on the metric of commit as the major contribution.

The bean plot of commit growth rate is drawn in figure 4. We further calculate the mean and median value of it. For r_c, it has a mean of 2,227 and a median of 125. The mean value is far greater than the median value, which indicates that a few ecosystems could have a very high commit growth rate, while other ecosystems' growth rates are around the median value. We sort and extract ecosystems whose commit growth rate is higher than the mean value of 2,227 and find six OSS ecosystems: (1) Microsoft, (2) Intel, (3) SAP, (4) Oracle, (5) Amazon, and (6) Tencent Holdings. From the above ecosystems, Microsoft has the highest commit growth rate among all ecosystems and its commit growth rate is almost six times as fast as the second top ecosystem (*i.e.*, Intel).

To sum up, in RQ1, we find 40 Global 500 companies that have built up their OSS ecosystems on GitHub. Commit growth rate is chosen as the major metric to measure contributions given to OSS *ecosystems. We find great difference between the mean and the median value of commit growth rate, indicating that most* OSS *ecosystems receive hundreds of commits per month, on average. But companies like Microsoft, Intel, SAP, Oracle, Amazon, and Tencent Holdings receive more than 2,000 contributions per month. For other companies who are seeking chances to develop an open-source environment, they can put an eye on the six heavily-contributed ecosystems to gain insights from them.*

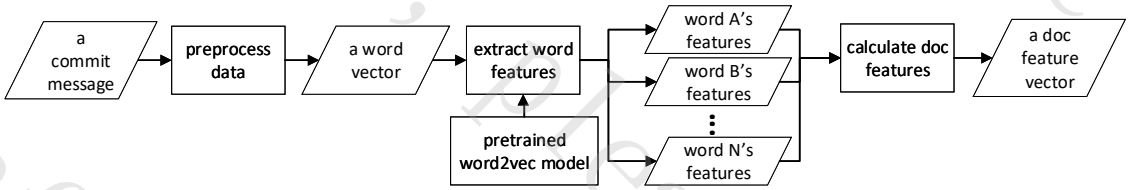

Figure 5: The workflow of feature extraction

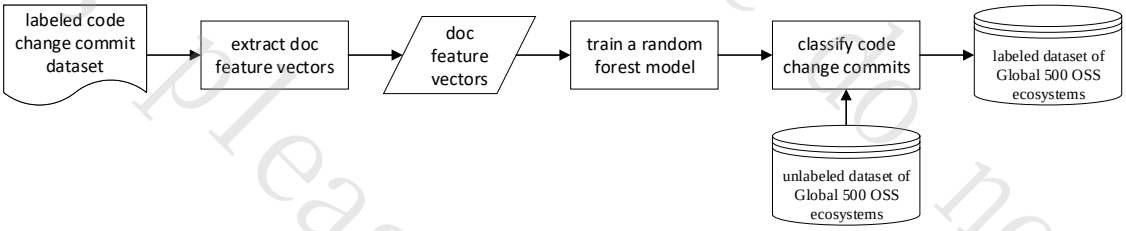

Figure 6: The workflow of classification model training

RQ2: What kinds of contributions and repositories reside in the OSS ecosystems of Global 500?

Motivation: In this RQ, we try to find out what kinds of repositories reside in the selected OSS ecosystems. Moreover, we aim to identify what kinds of contributions are given to these ecosystems. Repository and commit classification are the two major tasks here. Repositories can be categorized manually since there aren't too many of them. But we need to find an automatic classification method for commits since there are over 4 million commit messages, which is extremely time-consuming with manual labelling. With the classification result, we offer information about popular repository topics within and code change composition of the ecosystems. Practitioners can build their ecosystems by following the popular trends and use the latest techniques to avoid choosing outdated project topics. They can also understand how code changes are distributed in a mature ecosystem, helping them keep a clear mind as their ecosystems evolve.

Approach: First, we explore the types of repositories in our dataset. Since there is no existing category to classify the repositories, an exploratory study is carried out manually. We follow an open labeling process to assign multiple labels to each repository. The labels have no priority among each other. Sometimes, a GitHub repository will be assigned multiple tags by its owner, showing to which areas the repository belongs. If there are existing tags in a GitHub repository, then we pick a few tages that are representative to the repository's 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 occurrence frequency of all tags. Tags of similar meaning are merged to reduce triviality of the result.

After that, we study the type of code changes by examining the commit messages given to GitHub repositories of the Global 500. Following existing code change classification studies [24, 42], we also categorize commit messages into three types: *corrective*, *adaptive*, and *perfective*. The classification process is divided into three subsequent steps, that is, (1) training dataset preparation, (2) feature vector extraction, and (3) classification model training.

To train a ML model for commit messages classification, it is necessary to prepare a labelled dataset. Therefore, we utilize a publicly available dataset [24] of 1151 labelled commit messages that we modified to our needs. 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, which is not enough to train a ML model for our classification task on commit messages from over 2000 projects. Therefore, we extend the dataset to support our needs by adding into it more commit messages and keywords. With the population size of 4,032,041, confidence level of 95%, and 5% margin of error, a sample size of 385 is needed based on sample size determination formula. Therefore, we extract 400 commit messages for labeling purposes. The labeling process is done by two annotators (graduate students) who followed the software maintenance definition of Swanson [42], the classification schema of Amor et al. [2], and the labeling process of Levin and Yehudai [24]. Each code change commit is labeled with one of the three categories: (1) *corrective*, (2) *adaptive*, or (3) *perfective*. Commit messages with few textual information and unclear purpose were removed later on by the annotators. After the labeling, we use Cohen's Kappa to check the label's consistency of the two annotators who achieved 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 labeled commits are added into the original dataset that we use to train a commit

message classification model. We adopt feature vector extraction to turn our commit message strings into fixed-dimension feature vectors as shown in Figure 5. As a first step, we tokenize and break each commit message into individual linguistic units. 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. After the following steps, we keep words that match our keyword list and discard the ones that did not match. We calculate the occurrence frequency of all words that appeared in our dataset. Additional keywords are chosen and tested by an ablation study with various keyword combinations to find the best keyword list. Detailed experimental results are discussed in the following result part.

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 for computing vector representations of words. As shown in the Figure 5, 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 classification model's training process is shown in Figure 6. With the 1151+391 labeled 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 sent for training a random forest model.

Different keyword lists are used for training and testing to select a model that achieves the highest accuracy. After that, our 4,032,041 unlabeled commit messages are fed into our trained random forest model for classification in order to generate a labeled dataset of the Global 500 OSS ecosystems.

The 4,032,041 labeled commit messages are mapped back to their OSS ecosystems. The proportion of *corrective, adaptive,* and *perfective* commits are calculated for each ecosystem. Qualitative analysis is performed on the statistical result.

Table 2: Highest occurrence frequency tags

Framework	631	Management Tool	92
Cloud	333	IoT	92
Example	235	DataBase	90
Programming Language	191	Platform	83
Documentation	172	Command Line Interface	72
Application	143	Mobile	60
Container	138	Office	58
Artificial Intelligence	118	Network	56
Data Science	117	Operating System	54
DevOps	111	Build	51
Code Editor	101	Server	40
Code Analysis	101	Test	39

Results: For repository categorization, the open labeling process finds 102 tags in total. Considering some tags are of similar meaning, we manually double check and merge similar tags, resulting in 95 different categories. After sorting based on the tag occurrence frequency, tags whose occurrence frequencies are above the third quartile is shown in Table 2. Our results suggest that *framework* is the most popular tag in the Global 500 ecosystems. *Cloud service, programming language, artificial intelligence, data science,* and *IoT* are the top-5 application areas in OSS ecosystems. For software engineering, *container* and *DevOps* are two popular techiques being developed in ecosystems. Tags such as *example*, *documentation*, *build*, and *test* are regular software development components, which are not uncommon.

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 our model by achieving 0.675 on 10-cross validation accuracy. Thus, our extended keyword list contains 19 original keywords and 6 additional keywords.

The 40 ecosystems are sorted based on their commit numbers, gathered into four groups by first, second, and third quartiles (Q1, Q2, and Q3), and calculate the percentage of three types (see Figure 7). Among the 40 ecosystems, 90% of them have more perfective code change commits than any other types. The remaining 10% of ecosystems have adaptive code changes as the highest percentage. None of the ecosystem 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 new environment, such as cross-platform migration and hardware update, adaptive code changes also exist in these ecosystems to make the ecosystems evolve. However, for a few ecosystems, such as Oracle, Tencent Holdings, and General Electric, demonstrate differences in the code change distribution. Oracle has very high percentage in adaptive code changes. Tencent Holdings and General Electrics have most of the commits as perfective and very few corrective as well as adaptive code changes. The three special cases may indicates potential differences in the development mode of their ecosystems. Detailed reasons for these special cases are not delved in this study. We also find that the OSS ecosystems of Global 500 are actively attracting contributions to projects in the areas of *cloud service, programming language, artificial intelligence, data science,* and *IoT*.

Our results suggest that most of the contributions given to the Global 500 OSS *ecosystems are perfective and there are a few adaptive commits, while none of the ecosystem has the highest percentage of corrective code changes. In very limited cases, we find that some companies many follow a different development approach and have more adaptive commits.*

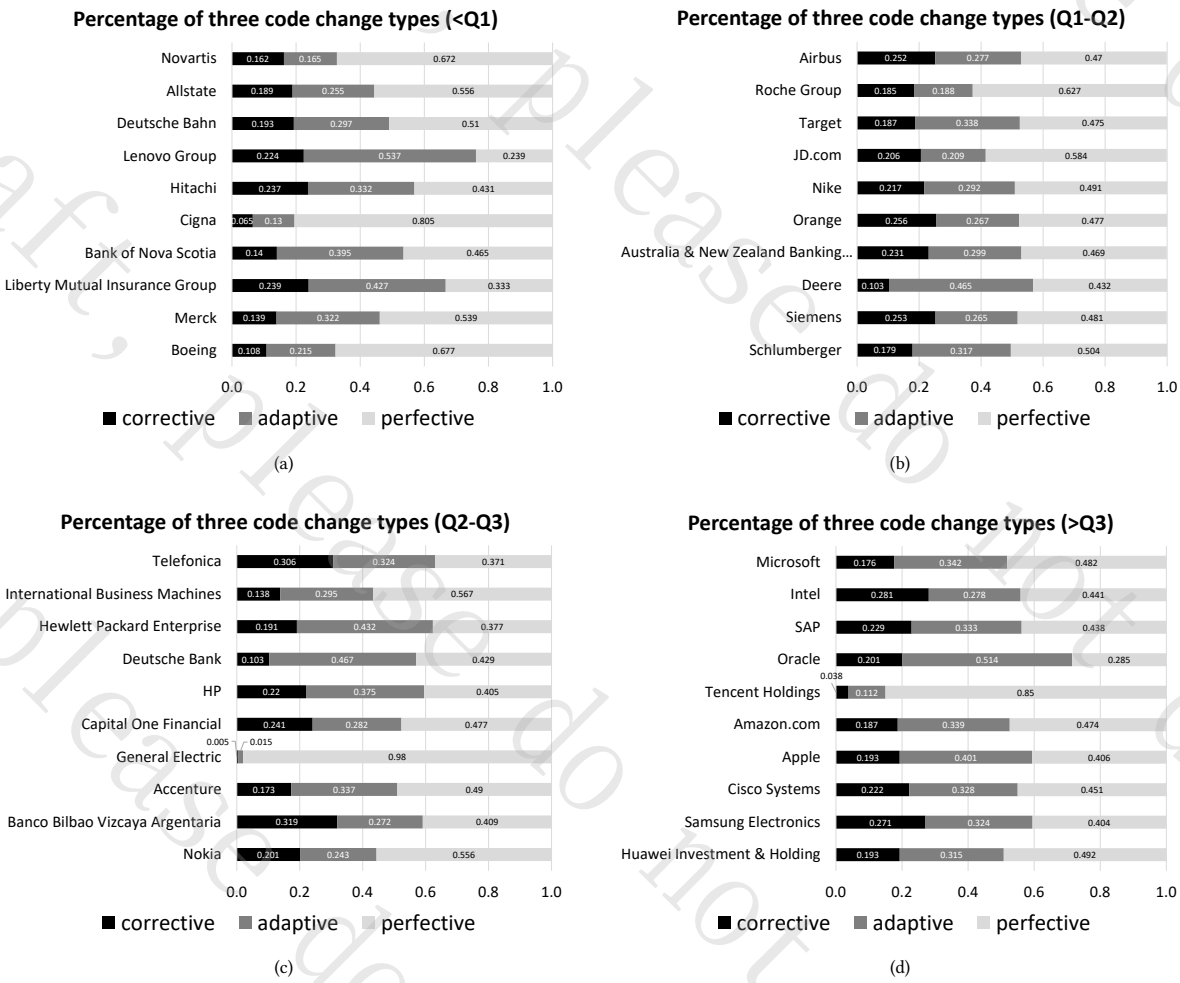

Figure 7: Code change percentages of perfective, corrective, and adaptive commits for 40 OSS ecosystems

RQ3: Who gives contributions to the OSS ecosystems of Global 500?

Motivation: In this RQ, we try to identify who contribute to the OSS ecosystems. It is necessary to give an exploration on the contributors and the roles they play in an OSS ecosystem development process. We try to understand 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 participation of external contributors to an ecosystem, the more active an OSS ecosystem. The results provide an outlook on what OSS ecosystems of influential companies look like, providing facts to other companies who are considering of building up their own OSS ecosystems.

Approach: To identify who contribute to the ecosytems, identity disambiguation is executed as a data preprocessing step because commit number cannot simply represent contributor number. Several code change commits could be from the same contributor. So we need to gather commits that come from the same entity into one group, counting them as from one instead of multiple contributor. We follow the method proposed by Amreen et al. [3] and build up a ML model for identity disambiguation. To train a robust model for id disambiguation, a total of 91,635 pieces of training data are randomly sampled and labelled independently by two graduate student. The sample size far exceeds the minimum sample size requirement of 385. The training set is extended 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. Various candidate features are tested with the training set and non-critical features are dropped. Eight features are selected to construct the feature vector for disambiguation, which is explained in detail as below. For any two piece of data, repository id and company email domains are compared to

Table 3: 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

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 time zone of a commit is also considered. The 24 hours within a day is divided into 24 time zones. We calculate the absolute value of time zone difference as another feature value. This feature is created because commits from the same entity are expected to be submitted at similar time period in a day. Afterwards, we train an RF model by using the above features and dataset, which achieved average accuracy of 99% with 10-fold cross validation. The model is further adopted in the identity disambiguation process on the whole dataset of 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 who are from outside of a company, we initially divide contributors into two high-level categories: internal and external. Internal contributors are those who submit commit 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 as an external contributor.

To detect internal contributors, we check to which company's OSS ecosystem a commit is given. The company's email domain is extracted to make a comparison with the contributor's email domain. If two domains match, then it indicates that the contributor makes the code change commit to his/her own company's ecosystem with the company's email. The contributor is considered as 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 Global 500 ecosystem 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 again, adopting card sort to both groups to divide each group into more detailed subgroups. This process is repeated until no new group is added into 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.

However, in some cases, an email domain ending with ".com" looks like from a commercial company, but is actually provided for personal use, like a personal email account. Therefore, we create a new label named individual contributors to distinguish between commercial and personal emails. A matching list [21] containing 3,782 domains of free email provider 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 like ".ca", ".cn", and etc. Another matching list [17] 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 labeled as others. Finally, contributors are clustered into nine groups as shown in Table 3.

Results: We divide each group's commit number by the total commit number and divide contributor number by the total contributor number to obtain each group's commit percentage and contributor percentage. The results are illustrated in Figure 8.

Own Global500 and *Individual* groups occupy the biggest portion of contributions for the selected OSS ecosystems with 41.6% and 40.0%, respectively. *Non500 Corporation* has the third highest proportion of contribution of 9.3%, while the remaining sectors contribute the rest 9.9% of the total contributions.

If we merely calculate the commit percentage without considering the identity duplication, the result shows a very different distribution. *Own Global500* occupies 73.4% of all the contributions. *Individual* group only has 12.3%, followed by *Non-profit Organization* with 4.9% and *Non500 Corporation* with 3.5%. By comparing the contributor percentage with commit percentage as shown in Figure 9, we find that 58.4% 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 73.4% of commits to the OSS ecosystems, while external developers contribute only 26.6% of commits.

Our results depict that external developers are actively participating in the OSS development of Global 500 ecosystems, but the code change contributions given by external developers are far less than internal employees (see Figure 8). 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

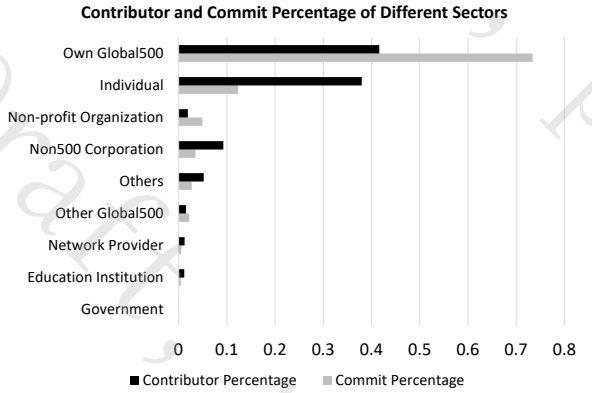

Figure 8: Distribution of contributor and commit based on different groups

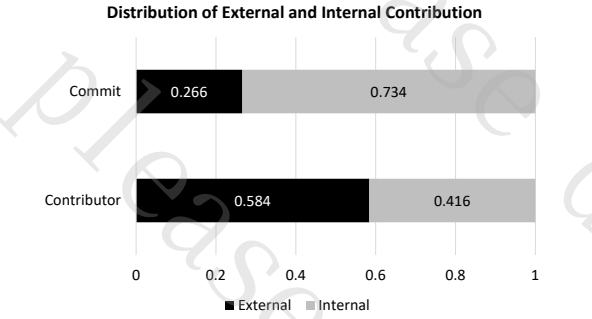

Figure 9: Commit number per contributor for each group

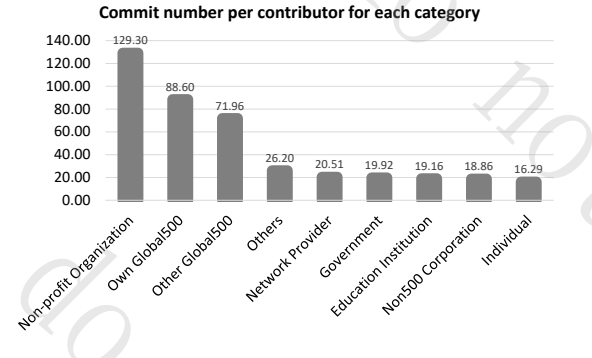

Figure 10: Distribution of external and internal contribution

by external developers are through 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 deviates from the internal development path and might be declined to keep development process consistent. Another possible reason is that external developers do not have that much time contributing to the ecosystems compared to the internal employees.

We also compare the per capita code change commits in each group as illustrated in Figure 10. Our results show that *Non-profit Organization* group has the highest per capita commit of 129.30, that is, commits per contributor. Moreover, *Own Global500* has the second and *Other Global500* has the third highest per capita commits of 88.60 and 71.96 respectively, while the remaining sectors have a similar per capita commits. We find that developers from the *Individual* group has the lowest per capita commit number of 16.29. 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 is 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 only occupy around one forth of total commit number, which is far less than internal contributors. To make an OSS ecosystem lively and attract more external developers, we recommend commercial companies—who are building up their own OSS ecosystem—to develop strategies to raise the code change acceptance rate of external, especially Individual developers outside the company.*

5 THREATS TO VALIDITY

Internal Validity: The internal threats to validity comes 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. However, since the annotators could have different level of software development experiences, the labeled commits could be at the potential risk 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. The RF model achieves a relatively high classification accuracy on the test set. But there is still one third probability to assign a wrong label to the commit messages. In the future, we plan to (1) include more annotators to ensure a more objective labelling process, (2) try more keyword combinations including the words behind top 30, and (3) improve the classification accuracy by choosing better machine learning models and adopt more feature dimensions.

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 draw 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.

6 RELATED WORKS

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

6.1 Studies related to OSS Ecosystems

Jansen et al. [20] study the technical and business aspects of software engineering in vibrant ecosystems. The authors point out that failing to develop a software in a software ecosystem has led to loss of (1) competition, (2) intellectual property, and (3) jobs in the software industry. In the study of Kilamo et al. [22], the authors discuss migrating an industrial software project from a closed to an OSS ecosystem An industrial case is introduced to find the (1) supporting processes, (2) guidelines, and (3) best practices. To find the analogy and difference between OSS ecosystems and ecological communities, Matragkas et al. [28] perform an empirical study by using GitHub ecosystems. Specifically, the authors 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 the members of large a project are passive users. Eckhardt et al. [10] study how meritocracy governance affects a project's health in the Eclipse OSS ecosystem. From the management perspective, the authors find that the ecosystem is not always fair and merits are beneficial in only some case. In a research by Zhang et al. [47], company participation and collaboration in the OpenStack ecosystem are studied by proposing a methodological framework. The authors characterize collaboration patterns and identify engagement strategies that companies adopt for participation. Such results may support a company in defining its own OSS strategy.

6.2 Studies related to code change classification

The early stage code change classification is proposed in the late 1970s by Swanson [42] and Lientz et al. [25]. The former study gives an exploration on the dimensionality of software maintenance and proposes three dimensions to it such as *corrective*, *adaptive*, and *perfective* maintenance. The latter study analyzes software maintenance categories by surveying maintenance managers. The authors conclude that the maintenance contains 17.4% corrective, 18.2% adaptive, 60.3% perfective, and 4.1% other effort. A roadmap on software maintenance and evolution suggested by Bennett and Rajlich [6] shows that around 75% of the maintenance efforts are *adaptive* and *perfective*, 21% are *corrective*, and the remaining 4% are categorized as a preventive work. More case studies are done by Schach et. al. [36]. They 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 survey-based study.

After the early exploratory stage, code change classification methods are developed to perform automatic categorization. Amor et al. [2] present a detailed classification schema for code changes baed on the analysis of textual descriptions attached to each transaction in the versioning system. Similarly, Hindle et al. [15, 16] study large commits and their classification method by mining software repositories. Likewise, other authors utilize Machine Learning (ML) techniques to classify code changes [18, 19, 24, 27, 34]. In particular, Levin and Yehudai [24] employ 1151 labelled commits from 11 popular OSS projects to classify code changes, while Sabetta and Bezzi [34] identify security-related commits through natural language processing. [18, 19] introduce code density as a novel feature to improve the classification model. [27] add three additional features and tests with XGoost algorithm.

6.3 Studies related to identity disambiguation

Identity disambiguation is a technique to link data records from the same entity together. Robles and Gonzalez-Barahona [33] study the problem where developers use different identities when interacting with various tools. The authors propose a heuristic-based approach to find which developers are using different identities. To resolve multiple email aliases, Bird et al. [7] use hand-inspection. Steorts et al. [41] compare multiple traditional blocking techniques which link records from the same entity and also discuss about privacy-concerned issues. In another study, authors build a methodology for id disambiguation among social and Q&A platforms such as Stack Overflow, GitHub, and Twitter [37], while Amreen et al. [3] use supervised learning to detect developer identity duplications from OSS repositories. Fry et al. [12] propose a dataset containing author IDs from VCS systems and a method to find all author IDs belonging to a single developer in the entire dataset. They also share the list of all author IDs that were found to have aliases.

7 CONCLUSIONS & FUTURE WORKS

In this research, we conduct an empirical study on the OSS ecosystems of the Global 500. Our analysis shows that 40 of the Global 500 companies have already built up their own OSS ecosystems on GitHub. The ecosystems of *Microsoft, Intel, SAP, Oracle, Amazon*, and *Tencent Holdings* have extremely high commit growth rate, which could indicate that these ecosystems are being actively developed and attract a large number of code change contributions. *Container, DevOps, cloud service, programming language, artificial intelligence, data science,* and *IoT* are popular OSS development topics of these influential companies, providing useful information on project selection for other companies who tend to built their own OSS ecosystems. We also find that 90% of the OSS ecosystems have more *perfective* than *adaptive* and *corrective* code changes, while the remaining 10% have the highest percentage of *adaptive* code changes in their ecosystems. *Corrective* code change never occur as the highest percentage in code changes and it occupies a stable proportion around 10-30% in all ecosystems. The result provides practitioners a view on the code change composition of mature OSS ecosystems Moreover, we show that more external developers than company employees participate in OSS ecosystems development. But external developers have less per capita commit contributions than internal developers. In particular, the *Individual* group has the most contributors but the least per capita commits among all external contributor groups. We suggest commercial companies to develop strategies to raise code change acceptance rate of external developers to maintain a lively OSS ecosystem.

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

REFERENCES

[1] Mark Aberdour. 2007. Achieving quality in open-source software. *IEEE software* 24, 1 (2007), 58–64.

[2] Juan Jose Amor, Gregorio Robles, Jesus M Gonzalez-Barahona, and Alvaro Navarro. 2006. Discriminating development activities in versioning systems: A case study. In *Proceedings PROMISE*, Vol. 2006. Citeseer, 2nd.

[3] Sadika Amreen, Audris Mockus, Russell Zaretzki, Christopher Bogart, and Yuxia Zhang. 2020. ALFAA: Active Learning Fingerprint based Anti-Aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering* 25, 2 (2020), 1136–1167.

[4] Morten Andersen-Gott, Gheorghita Ghinea, and Bendik Bygstad. 2012. Why do commercial companies contribute to open source software? *International journal of information management* 32, 2 (2012), 106–117.

[5] Stephanos Androutsellis-Theotokis, Diomides Spinellis, Maria Kechagia, Georgios Gousios, et al. 2011. Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management* 4, 3-4 (2011), 187–347.

[6] Keith H Bennett and Václav T Rajlich. 2000. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*. 73–87.

[7] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*. 137–143.

[8] Andrea Bonaccorsi and Cristina Rossi. 2003. Why open source software can succeed. *Research policy* 32, 7 (2003), 1243–1258.

[9] Jailton Coelho and Marco Tulio Valente. 2017. Why modern open source projects fail. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*. 186–196.

[10] Evert Eckhardt, Erwin Kaats, Slinger Jansen, and Carina Alves. 2014. The merits of a meritocracy in open source software ecosystems. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*. 1–6.

[11] Facebook. 2021. Facebook Open Source. https://opensource.fb.com/

[12] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2020. 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*. 518–522.

[13] Alfonso Fuggetta. 2003. Open source software—an evaluation. *Journal of Systems and software* 66, 1 (2003), 77–90.

[14] Google. 2020. Google Open Source. https://opensource.google/

[15] Abram Hindle, Daniel M German, Michael W Godfrey, and Richard C Holt. 2009. Automatic classication of large changes into maintenance categories. In *2009 IEEE 17th International Conference on Program Comprehension*. IEEE, 30–39.

[16] Abram Hindle, Daniel M German, and Ric Holt. 2008. What do large commits tell us? A taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*. 99–108.

[17] Hipo. 2021. University Domains and Names Data List & API. https://github.com/Hipo/university-domains-list

[18] Sebastian Hönel, Morgan Ericsson, Welf Löwe, and Anna Wingkvist. 2019. 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, 109–120.

[19] Sebastian Hönel, Morgan Ericsson, Welf Löwe, and Anna Wingkvist. 2020. Using source code density to improve the accuracy of automatic commit classification into maintenance activities. *Journal of Systems and Software* 168 (2020), 110673.

[20] Slinger Jansen, Anthony Finkelstein, and Sjaak Brinkkemper. 2009. A sense of community: A research agenda for software ecosystems. In *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 187–190.

[21] T. Brian Jones. 2021. tbrianjones/free_email_provider_domains.txt. https://gist.github.com/tbrianjones/5992856

[22] Terhi Kilamo, Imed Hammouda, Tommi Mikkonen, and Timo Aaltonen. 2012. From proprietary to open source—Growing an open source ecosystem. *Journal of Systems and Software* 85, 7 (2012), 1467–1478.

[23] Pavneet Singh Kochhar, Eirini Kalliamvakou, Nachiappan Nagappan, Thomas Zimmermann, and Christian Bird. 2019. Moving from closed to open source: observations from six transitioned projects to github. *IEEE Transactions on Software Engineering* (2019).

[24] Stanislav Levin and Amiram Yehudai. 2017. 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*. 97–106.

[25] Bennet P Lientz, E. Burton Swanson, and Gail E Tompkins. 1978. Characteristics of application software maintenance. *Commun. ACM* 21, 6 (1978), 466–471.

[26] Johan Linåker, Patrick Rempel, Björn Regnell, and Patrick Mäder. 2016. 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, 63–81.

[27] Richard VR Mariano, Geanderson E dos Santos, Markos V de Almeida, and Wladmir C Brandão. 2019. Feature changes in source code for commit classification into maintenance activities. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 515–518.

[28] Nicholas Matragkas, James R Williams, Dimitris S Kolovos, and Richard F Paige. 2014. Analysing the 'biodiversity' of open source ecosystems: the GitHub case. In *Proceedings of the 11th working conference on mining software repositories*. 356–359.

[29] Microsoft. 2021. Microsoft Open Source. https://opensource.microsoft.com/

[30] Vishal Midha and Prashant Palvia. 2012. Factors affecting the success of Open Source Software. *Journal of Systems and Software* 85, 4 (2012), 895–905.

[31] Tim O'Reilly. 1999. Lessons from open-source software development. *Commun. ACM* 42, 4 (1999), 32–37.

[32] James W Paulson, Giancarlo Succi, and Armin Eberlein. 2004. An empirical study of open-source and closed-source software products. *IEEE transactions on software engineering* 30, 4 (2004), 246–256.

[33] Gregorio Robles and Jesus M Gonzalez-Barahona. 2005. Developer identification methods for integrated data from various sources. *ACM SIGSOFT Software Engineering Notes* 30, 4 (2005), 1–5.

[34] Antonino Sabetta and Michele Bezzi. 2018. A practical approach to the automatic classification of security-relevant commits. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 579–582.

[35] Walt Scacchi, Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani. 2006. Understanding free/open source software development processes.

[36] Stephen R Schach, BO Jin, Liguo Yu, Gillian Z Heller, and Jeff Offutt. 2003. Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering* 8, 4 (2003), 351–365.

[37] Giuseppe Silvestri, Jie Yang, Alessandro Bozzon, and Andrea Tagarelli. 2015. Linking Accounts across Social Networks: the Case of StackOverflow, Github and Twitter.. In *KDWeb*. 41–52.

[38] Param Vir Singh. 2010. 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)* 20, 2 (2010), 1–27.

[39] Diomidis Spinellis, Georgios Gousios, Vassilios Karakoidas, Panagiotis Louridas, Paul J Adams, Ioannis Samoladas, and Ioannis Stamelos. 2009. Evaluating the quality of open source software. *Electronic Notes in Theoretical Computer Science* 233 (2009), 5–28.

[40] Diomidis Spinellis, Zoe Kotti, Konstantinos Kravvaritis, Georgios Theodorou, and Panos Louridas. 2020. A Dataset of Enterprise-Driven Open Source Software. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 533–537.

[41] Rebecca C Steorts, Samuel L Ventura, Mauricio Sadinle, and Stephen E Fienberg. 2014. A comparison of blocking methods for record linkage. In *International conference on privacy in statistical databases*. Springer, 253–268.

[42] E Burton Swanson. 1976. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering*. 492–497.

[43] Kris Ven, Jan Verelst, and Herwig Mannaert. 2008. Should you adopt open source software? *IEEE software* 25, 3 (2008), 54–59.

[44] Georg Von Krogh and Eric Von Hippel. 2006. The promise of research on open source software. *Management science* 52, 7 (2006), 975–983.

[45] Joel West and Scott Gallagher. 2006. Challenges of open innovation: the paradox of firm investment in open-source software. *R&d Management* 36, 3 (2006), 319–331.

[46] Ming-Wei Wu and Ying-Dar Lin. 2001. Open source software development: an overview. *Computer* 34, 6 (2001), 33–38.

[47] Yuxia Zhang, Minghui Zhou, Klaas-Jan Stol, Jianyu Wu, and Zhi Jin. 2020. 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, 1196–1208.