# Abstract

# Introduction

# Data Collection

To conduct our analysis and test hypotheses, we use the public historical data from GitHub Archive. GitHub Archive comprises the data and content of more than 2.8 million open source GitHub repositories and is hosted on Google BigQuery platform (Hoffa 2016). The data is stored in a hybrid database format that combines structured and unstructured data. Google BigQuery platform infrastructure allows SQL-like queries to access the entire dataset and other non-SQL requests (BigQuery 2023).

GitHub provides over 20 event types, ranging from new commits and fork events, to opening new issue tickets, commenting, and adding members to a project (GitHub 2023). We focus on activities that are most commonly used by the software developers to assess the quality and make decisions whether to join the open-source project (Dabbish et al. 2012): commits, pull requests, issues, forks, and watches. In addition, we extract the details about each event from the payload string available in the GitHub Archive database. A “payload” string field contains the JSON encoded activity description with detailed information about the content of the changes made, previous version of the file, comments the developer attached to the pull request or commit, and whether the pull request or issue were closed. To access the json-stored information, we use json\_extract() function.

Google BigQuery GitHub dataset contains data beginning 2011. However, activity archives for dates between 2/12/2011-12/31/2014 was recorded from the Timeline API that has been replaced by Events API beginning 1/1/2015. The change in API changed the structure of data and created the discrepancy in records’ schemas before and after 2015. To alleviate the discrepancy and unify the data structure for the proposed analysis, we extracted the missing information from the payload fields in tables before 2015. Due to the Ethereum project OSS development has started in late 2013, over one year of data has been extracted using the alternated query and processed to match the rest of the dataset.

In this study, we focus on the OSS activities inside the Ethereum community. The main webpage of the Ethereum software development is <https://github.com/ethereum>. The structure of the GitHub platform allow creation of different repositories, i.e., folders, within the main Ethereum webpage. The addresses of the repositories begin with <https://github.com/ethereum>/ <repository name>. Therefore, to filter data in the GitHub Archive related to Ethereum development, we simply extract all activities associated with all repositories that with <https://github.com/ethereum>/ <repository name>. Note that the GitHub functionality allows other users to create copies of the repositories and save these copied folders within their webpages. Such process is called forking and the copied repositories are referred to as forks. Though the copied repositories will have the ‘<user webpage>/ethereum/’ in the name of their repository, we do not collect data for the activities in forks.

We collect data from December 2013 – May 2023 and aggregate them at a repository –actor.login – day level. Overall, we collect 460,000 records after processing 19.08 TB of data. Using the initial dataset, we further aggregate the data at a repository – day level to test our hypotheses.

# Hypothesis Testing

## Hypothesis 1

H1: The roles of core versus peripheral developers: *Greater participation of peripheral developers in open-source projects are associated with greater awareness about open-source products* (Setia et al. 2012).

To test our first hypothesis, we first need to organize our dataset and construct the necessary variables. The main independent variable is the number of peripheral developers. The developers who contribute to the repository can be classified into two types: core and peripheral. The core developer is the one who is heavily involved in the project development, typically has the right to apply proposed changes to the main software file, or merge pull requests, and strategically decide the future functionality of the software product. Based on the literature, core developer has been defined as contributing more than 12% of code to the project (Mockus et al. 2002). We adopted this threshold and marked as core developers those who contributed 12% or more of total activities in a repository over the past 30 days. All other developers were marked as peripheral. We also checked for the abnormal patterns of contributions by unique contributors, e.g., one across all repositories on a daily basis, and ran our analysis without them. Typically, such activity is associated with bots and should not be considered in our analysis. Moreover, the login name of such actors contains the ‘bot’ part, and we mark them as ‘is\_bot’ in our dataset. The results with and without bot activities are robust.

As the main dependent variable, we choose different measures of the project awareness quoted in the previous literature. One measure of the project awareness is number of forks. Developers tend to fork the repositories they are interested in for two reasons: (1) they want to contribute to the development and improvement of the software, and (2) they want to use the existing code for the development of their own project. In both cases the motivation to fork is rooted in the inherent quality of the software and its recognition by developer. In addition, the technology diffusion happens by means of forks, and the wider population becomes aware of it. The second measure of the project awareness is the number of watches. Watching a repository is similar to following the account on the social media as it enables to receive the notifications about any changes and updates applied to the watched account. Therefore, number of watches on GitHub is the direct measure of the popularity of the software project.

To test our first hypothesis, we run a panel fixed effects regression:

(1)

In equation (1), i – is the index of a repository within the Ethereum project, t – index of a time period, is the repository fixed effect that accounts for unique attributes of the repository that are not captured by other variables, – idiosyncratic error. As a measure of product awareness (), we use the number of watches (Watches) and the number of repository forks (Forks). The main independent variable is the number of peripheral developers (Peripheral). We also control for the number of core developers (Core) and the lifetime of the project since its initiation (Duration).

After we estimate the regression, our results confirm that the tested hypothesis: Number of peripheral developers is positively associated with the repository awareness (number of forks and number of watches). Addition of one peripheral developer increases the number of repository forks by 0.16 and number of watches by 0.52 on a daily basis. The results are presented in Table 1.

**Table 1. H1 Testing Results**

|  |  |  |
| --- | --- | --- |
| Variable | (1)  FE, DV=Forks | (2)  FE, DV=Watches |
| Peripheral | 0.1623\*\*\* | 0.515\*\*\* |
|  | (0.0027) | (0.0071) |
| Core | -0.0042\*\* | -0.055\*\*\* |
|  | (0.0016) | (0.0107) |
| Duration | 0.0003\*\*\* | 0.0001\*\* |
|  | (0.00002) | (0.00002) |
| Repository FE | Y | Y |
| Num obs | 101,081 | 101,081 |
| Adj R-sq | 0.646 | 0.833 |

**Note.** Standard errors in parentheses; + p<0.1, \* p<0.05, \*\* p<0.01, \*\*\* p<0.001

The result in Table 1 signify that peripheral developers are the drivers of the social diffusion in the collaborative networks like OSS community. The awareness about the project, or the repository in our case, is increasing as there are more developers who show interest in it but not yet heavily involved in the development activity (less than 12% of all activities in the past 30 days). Peripheral developers are involved in many other projects on the platform, therefore, when interaction with the focal repository is reflected on their profile, their collaborators are exposed to the focal project. The attention to the focal project can be reflected by subscribing for the repository updates (watching) or forking the code for future modification or use. While watching serves as a powerful signal of popularity of the code, number of forks signify the social recognition and impact (Petryk et al. 2023).

## Hypothesis 2

Due to the importance of the social connections in the OSS community for the information diffusion, we further investigate their impact with the next hypotheses.

H2: The nuanced relationship between developer repeated collaboration and project success: A moderate level of internal cohesion within a project is better for a project’s success than very high or very low levels of internal cohesion (Singh et al. 2011).

To test our second hypothesis, we construct the measure of internal cohesion following Singh et al. (2011). The presence of repeated collaborations among project developers is related to strong interpersonal connections (Uzzi 1997). We calculate the number of developer pairs from the focal repository that worked on other repositories within Ethereum project and divide it over the total number of pairs that exist in the focal repository to calculate the number of repeated ties, or the internal cohesion metric. The resulting metric is the main independent variable. The main dependent variable is a project’s success. Subsequent to the extant literature, we consider a project’s successas a project’s rate of knowledge creation and measure it as number of commits (Boh et al. 2007, Crowston et al. 2003). We also control for the the lifetime of the project since its initiation (Duration).

To test our second hypothesis, we run a panel fixed effects regression:

(2)

In equation (2), i – is the index of a repository within the Ethereum project, t – index of a time period, is the repository fixed effect that accounts for unique attributes of the repository that are not captured by other variables, – idiosyncratic error. include the lifetime of the project since its initiation ().

The results are presented in Table 2.

**Table 2. H2 Testing Results**

|  |  |  |
| --- | --- | --- |
| Variable | (1)  OLS, DV=lnCommits | (2)  RE, DV= lnCommits |
| mcIntCohesion | -0.002\*\* | -0.004 |
|  | (0.002) | (0.005) |
| mcIntCohesion^2 | -0.000004 | 0.0001 |
|  | (0.00001) | (0.00006) |
| Duration | -0.0001\*\*\* | -0.0001\*\*\* |
|  | (0.000003) | (0.00004) |
| Intercept | 0.660\*\*\* | 0.452\*\*\* |
|  | (0.004) | (0.026) |
| Num obs | 101,080 | 101,080 |
| Adj R-sq | 0.014 | 0.016 |

**Note.** Heteroskedasticity-consistent and autocorrelation-corrected standard errors in parentheses; + p<0.1, \* p<0.05, \*\* p<0.01, \*\*\* p<0.001

## Hypothesis 3

Building on the prior evidence of the relationship between project popularity and structure of contribution, we point our attention to the study by Medappa and Srivastava (2019) that suggests the nonlinear relationship between the superposition of the project and its popularity. We follow with the third hypothesis:

H3: The nuanced relationship between project structure and popularity: *The degree of superposition, that is, the ratio between the total number of versions of the OSS project to the total number of individual contributions to the project, has a nonlinear relationship with project attractiveness in the OSS community: A moderate degree of project superposition is better for a project’s popularity than very high or very low levels of superposition* (Medappa and Srivastava 2019).

To test our third hypothesis, we construct the measure of superposition as the main independent variable. We calculate the number of developers who contribute to the repository over a time period and divide it over a number of releases made within the same time frame. The resulting metric is distributed between 0 and 1 by definition. If degree of superposition equals exactly 1, all of the releases were implemented by individual developers and added sequentially. The degree of superposition decreases as a project adopts a concurrent development approach and approaches 0 as a greater number of individual contributions get accumulated into individual releases (versions) of the project. Since our dependent variable is *project’s popularity* and is congruent with the dependent variable in our first hypothesis, we also control for the number of core and peripheral developers in our analysis.

We run a panel fixed effects regression (3) to test our third hypothesis:

(3)

In equation (3), i – is the index of a repository within the Ethereum project, t – index of a time period, is the repository fixed effect that accounts for unique attributes of the repository that are not captured by other variables, – idiosyncratic error. include the number of peripheral developers (), the number of core developers (), and the lifetime of the project since its initiation ().

The results are presented in Table 3.

**Table 3. H3 Testing Results**

|  |  |  |
| --- | --- | --- |
| Variable | (1)  FE, DV= Watches | (2)  FE, DV=Forks |
| Superposition | 14.967\*\*\* | 6.412\* |
|  | (3.727) | (2.757) |
| Superposition^2 | -19.145\*\*\* | -7.903\* |
|  | (5.790) | (3.550) |
| Peripheral | 0.487\*\*\* | 0.206\*\*\* |
|  | (0.053) | (0.024) |
| Core | 0.002 | 0.002 |
|  | (0.005) | (0.002) |
| Duration | -0.0004 | 0.0001 |
|  | (0.00004) | (0.0002) |
| Repository FE | Y | Y |
| Num obs | 1,593 | 1,593 |
| Adj R-sq | 0.756 | 0.645 |

**Note.** Heteroskedasticity-consistent and autocorrelation-corrected standard errors in parentheses; + p<0.1, \* p<0.05, \*\* p<0.01, \*\*\* p<0.001

The result in Table 3 signifies that there is a non-linear relationship between superposition and project popularity ( is significant). As the level of superposition in the project increases, the project popularity increases but to a certain point. After a certain point, the further increase of superposition the project popularity starts decreasing ( is negative). Low superposition signifies that project releases occur after large number of contributions are accumulated. Such “productive deferral” makes the project less attractive to the developers due to lack of autonomy and independence in completing complex tasks. Hence, the developers are reluctant to follow such projects (Ryan and Deci 2000). High superposition signifies the high level of work independence that satisfies the autonomy need. However, the lack of collaboration and exchange of ideas creates the negative affective state and deters the interest to the project (Medappa and Srivastava 2019). Developers need the decent amount of challenge and autonomy to be interested in contributing to the project and opportunities to work with other developers on more complex tasks (Ke and Zhang 2010).

## Hypothesis 4

Due to the importance of the social connections in the OSS community for the information diffusion, we further investigate their impact with the next hypotheses.

H4: The importance of a project’s embeddedness among other OSS projects: *A project’s visibility and embeddedness in the global OSS community is positively associated with the project success.* (Grewal et al. 2006)

To test our fourth hypothesis, we construct the network of Ethereum repositories where the edges are established by the common collaborators. We use the R package igraph to do the computations. Based on the network, we calculate the centrality measures: degree centrality (), betweenness (), and eigenvector centrality () – to include as the main independent variables in our regression. We measure *project’s success* as the number of commits (Grewal et al. 2006) and number of forks and include them as the main dependent variables in our analysis. We also control for the lifetime of the project.

We run a panel fixed effects regression (4) to test our fourth hypothesis:

(4)

In equation (4), *i* – is the index of a repository within the Ethereum project, is the constant term that accounts for factors that affect the project success metrics not captured by other variables, – idiosyncratic error. As a measure of product success (), we use a metric of contribution – a number of commits (Commits) and a metric of diffusion – a number of repository forks (Forks). are the coefficients of interest and measure the effects of centrality measures on the project technical success. include the lifetime of the project since its initiation (). The results are presented in Table 4.

**Table 4. H4 Testing Results**

|  |  |  |
| --- | --- | --- |
| Variable | (1)  OLS, DV= lnCommits | (2)  OLS, DV= lnForks |
| Degree | 0.011\*\*\* | 0.007\*\*\* |
|  | (0.002) | (0.001) |
| Eigenvector | 9.112\*\*\* | 7.094\*\*\* |
|  | (2.238) | (1.391) |
| Betweenness | -0.001\*\*\* | -0.001\*\* |
|  | (0.0004) | (0.0002) |
| Duration | -0.0004\*\* | 0.001\*\*\* |
|  | (0.0002) | (0.0001) |
| Intercept | 0.769\*\* | 0.098 |
|  | (0.310) | (0.161) |
| Num obs | 400 | 400 |
| Adj R-sq | 0.319 | 0.713 |

**Note.** Heteroskedasticity-consistent and autocorrelation-corrected standard errors in parentheses; + p<0.1, \* p<0.05, \*\* p<0.01, \*\*\* p<0.001

The result in Table 4 signifies that there is a statistically significant relationship between a network structure and project success. As the degree centrality increases, the projects become more interconnected with others by the common developers that attracts more contributions ( in column 1) and the technological innovation produced in the focal project is being reused by many other developers ( in column 2). As the eigenvector centrality increases, the projects become more central in the network that turns them into magnets with respect to more contributions ( in column 1) and the diffusion of innovation ( in column 2). Lastly, As the betweenness centrality increases, the flow of talent becomes easier due to the project’s connectivity to other repositories that leads to reduction in contributions to the focal project ( in column 1) and the project forks ( in column 2).

# Results

TBD

# Reference List

1. Hoffa F. 2016. GitHub on BigQuery: Analyze all the open source code. <https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code>
2. BigQuery 2023. <https://cloud.google.com/bigquery/>
3. GitHub 2023. <https://docs.github.com/en/webhooks-and-events/webhooks/webhook-events-and-payloads>
4. Mockus, A., R. T. Fielding, J. D. Herbsleb. 2002. Two case studies ofopen source software development: Apache and Mozilla.ACMTrans. Software Engrg. Methodology11(3) 309–346.
5. Petryk, M., Qiu, L., and Pathak, P. 2023. The Impact of Open-Source Community on Cryptocurrency Market Price: An Empirical Investigation. MISQ, Forthcoming.
6. Uzzi, В. 1997. "Social Structure and Competition in Interfirm Networks: The Problem of Embeddedness," Administrative Science Quarterly (42:1), pp. 35-67.
7. Ke W, Zhang P (2010) The effects of extrinsic motivations and satisfaction in open-source software development. J. Assoc. Inform. Systems 11(12):784–808.
8. Ryan RM, Deci EL (2000) Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. Amer. Psych. 55(February):68–78.
9. Crowston, K., H. Annabi, J. Howison. 2003. Defining open source software project success. Proc. Internat. Conf. Inform. Systems, ICIS, Seattle, WA, Association for Information Systems, Atlanta, GA.
10. Boh, W., Slaughter, S., and Espinosa, J. 2007. "Learning from Experience in Software Development: A Multilevel Analysis," Management Science (53:8), pp. 1

**Appendix**

**Table A1. Data Structure for Analysis**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Repository | Date | User login | Is core (1/0) | Is bot  (1/0) | Num activities |
| A | Date1 |  |  |  |  |
| … | … |  |  |  |  |
|  |  |  |  |  |  |
| A | DateN |  |  |  |  |
| B | Date1 |  |  |  |  |
| … | … |  |  |  |  |
|  |  |  |  |  |  |
| B | DateN |  |  |  |  |

**Table A2. Auxiliary Data Structure to Compute Is\_Core Developer Variable**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Row | Repository | Date | User login | Num activities | Rolling count of activities, last 30 days |
| 1 | ethereum/alethzero | 2015-08-17 | 138296 | 1 | 31 |
| 2 | ethereum/alethzero | 2015-08-17 | 138296 | 1 | 32 |
| 3 | ethereum/alethzero | 2015-08-17 | 138296 | 1 | 32 |
| 4 | ethereum/alethzero | 2015-08-17 | 138296 | 1 | 30 |
| 5 | ethereum/alethzero | 2015-08-17 | 138296 | 1 | 30 |

# Appendix

# All Events

You will need access to [BigQuery](https://cloud.google.com/bigquery), a data warehouse created by Google to manage and analyze data, and [GH Archive](https://www.gharchive.org/), a project that records and archives the public GitHub timeline.

## 2015 to Present

On BigQuery, run the following script: <https://docs.google.com/document/d/1EgzNXdJjzMDhnUz9BjEuB-uMK0V9h5Z47QJSQcJSVnY/edit?usp=sharing>.

This script calculates and returns the frequency of twenty-eight different events: num\_activities, num\_dist\_commits, num\_dist\_commitcomments, num\_actors\_pushevents, num\_actors\_pusheventscomment, num\_dist\_pullreqopened, num\_dist\_pullreqclosed, num\_dist\_pullreqAll, num\_dist\_pullreqcomments, num\_actors\_pullreq, num\_actors\_pullreqcomment, num\_actors\_pullreq\_opened, num\_actors\_pullreq\_closed, num\_dist\_issuesopened, num\_dist\_issuesclosed, num\_dist\_issuesAll, num\_dist\_issuecomments, num\_actors\_issues, num\_actors\_issuescomment, num\_actors\_allevents, num\_actors\_issues\_opened, num\_actors\_issues\_closed, num\_forks\_event, num\_actors\_forks, num\_watch\_event, num\_actors\_watch, num\_releases, release\_payload.

BigQuery should return a table with entries that specify repositories, dates, actor IDs, actor logins, and the different recorded events for the year 2015.

Save the results as a local CSV file. Repeat the script for 2016, 2017, 2018, 2019, 2020, 2021, 2022, and 2023 by changing the line “FROM (SELECT \* FROM `githubarchive.month.\*` WHERE \_TABLE\_SUFFIX BETWEEN '201501' AND '201512') t1 WHERE t1.repo.name LIKE 'ethereum/%'” accordingly. Save each result as a CSV file.

## 2013 and 2014

Due to information shortages, we will begin to work locally on BigQuery, instead of relying on GitHub Archive. Prior to 2015, only actor logins were used to identify actors. Actor IDs had not been assigned to users. This resulted in slightly different changes in the scripts we used.

\*mariia explain how you retrieved the individual data files for each month of 2013 and 2014\*

On any IDE, such as Visual Studio Code, run the following Python script:

<https://docs.google.com/document/d/1XePirIttPQVDBOA5PNg7YpgknRt947mQkitOEz4IyB4/edit?usp=sharing>.

This script combines each CSV file into one called AllData20132014.csv.

\*mariia used her own SQL script to generate another merged file maybe have that instead\*

After extracting 2013-2014 data from GH Archive on BigQuery, we extract fields from json strings to match the format of our 2015-2023 data. First, we extract the repository name (repo\_name) from JSON repo field. Next, we extract the actor login (actor\_login) from JSON actor field. The Python script to retrieve the listed fields is accessible via [link](https://www.dropbox.com/scl/fi/dr1qnwd6s4xnvgyky5cho/Data-2013_2014_Analysis-To-Share-Clean-Copy.py?rlkey=irahziqp1frhe40yugsk3r2gx&dl=0). In case we notice that some of the information is left blank, we parse the payload field to extract the missing data.

On BigQuery, click add to upload a local file. Select AllData20132014.csv and a dataset to create the table. Then, run the following script: <https://docs.google.com/document/d/1at-KWEDrOUjcGVi8TT8X3ndWlIEPeA4fP4e1ztx6Yj8/edit?usp=sharing>.

Replace this line, from ethereum-project-383415.Data.Data20132014Merged, with your project and dataset name. My project is named ethereum-project-383415 and my dataset is named Data. This can vary, depending on user preferences.

This should generate twenty-eight different events for the years 2013 and 2014. Save the result as a CSV file.

# Core Users

## 2015 to Present

To test our first hypothesis, run the following Python script using an IDE: <https://docs.google.com/document/d/1ZHnmQ78TM00MtUnYASRu1zEaS27ve95XuVL7YlXi97k/edit?usp=sharing>.

Change the following lines: df = pd.read\_csv('Data2015.csv', low\_memory = False), merged\_df[columns\_to\_display].to\_csv('Data2015CoreUser.csv', index = False) to match the corresponding year. Repeat the script for 2016, 2017, 2018, 2019, 2020, 2021, 2022, and 2023. A file containing the results for each year should be generated in the chosen directory.

## 2013 and 2014

In the CSV file that contains all the events for 2013 and 2014, create an empty column called actor\_id to the right of date1 and to the left of actor\_login.

Now, run the following Python script using an IDE: <https://docs.google.com/document/d/18EIGAKRlG2rcNQChECNuSr52YKeBc5bCjYg3vyx2ibw/edit?usp=sharing>.

actor\_login is used as the unique identifier instead of actor\_id.