2021 IEEE Technology & Engineering Management Conference - Europe (TEMSCON-EUR) | 978-1-6654-4091-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/TEMSCON-EUR52034.2021.9488616

Analysis of the Open Source Software Development Project Properties and Practices

Ivan Čeh
Faculty of Mechanical Engineering and Naval Architecture
University of Zagreb
Zagreb, Croatia
ivan.ceh@fsb.hr

Goran Delač
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia
goran.delac@fer.hr

Mario Štorga
Faculty of Mechanical Engineering and Naval Architecture
University of Zagreb
Zagreb, Croatia
mario.storga@fsb.

***Abstract*—Open source software development is increasingly being used as an alternative method to traditional proprietary software development. The development of different version control systems and their hosts has further popularized this way of creating software. It has many distinctive properties from other types of software development including high number of participants and distribution of management which is why analysis of its dynamics can be useful to improve understanding of different factors and their effect on the process.**

Keywords—product development, open source software, GitHub

I. Introduction

The term "open source software development" was created at the end of the 20th century, to better emphasize its difference from the term "free software". It usually refers to the software released under a license that permits the inspection, use, modification and redistribution of the software's source code. [1] It can be developed in the similar way as closed source, proprietary software but often involves much bigger number of geographically and organizationally distributed developers. The motive of developers can vary, they can be paid for their contributions to the project or voluntary participants with different goals as making software free and accessible, development and display of their skills etc. Some of the most known open source projects are Linux operating system, Apache Web Server, internet browser Mozilla Firefox, programming language Python and many other programs of various types and uses. Besides individuals, many open source projects attract large commercial support of companies that use open source software in their business model. [2] It was estimated that in 2005 87% of U.S. companies and government institutions used some open source software as a part of their business process [3] and this number continues to grow.

The term "open innovation" was coined by Henry W. Chesbrough as "the use of purposive inflows and outflows of knowledge to accelerate internal innovation, and expand the market for external use of innovation, respectively" [3]. Both terms refer to growing paradigms with many similarities such as search for collaborations outside of company boundaries and exploitation of external knowledge. Some other typical common properties are geographical distance of collaborators which entails specific tools for distant communication and collaboration. These terms have some important differences, for example open innovation deals with different types of products contrary to open source development which always refers to the software. Besides that, the term open innovation is more flexible when it comes to the part of the process where it is applied. Open source refers exclusively to the development phase while open innovation is applied in other stages of the process including search for ideas, innovation of the business model and finding external paths to market. [2] Open source software development is also not always open innovation since the development process can still be limited to the company regardless of public source code disclosure. However, most of the big open source software projects fall into the category of open innovation. [4]

West and Gallagher [5] described some challenges of open source software and reasons why firms would invest in it if its results can be exploited by rival firms. Three main challenges mentioned are maximization of returns to internal innovation, incorporation of external knowledge into the firm's innovation activities and keeping external sources of innovation motivated to continue developing them. Some ways for a firm to benefit from open source software are creating innovations for internal and external commercialization, building absorptive capacity for external innovations and indirect profit from spillover or sale of related goods and products. Besides direct financial investment in the product, motivation to develop the product can be increased by customers who share their ideas to improve product quality. Before firms decide to contribute to a product, economic viability is useful to assess by considering spillovers to competitors.

The paper is structured as follows. Section II discusses some existing papers related to data analysis of GitHub repositories and their properties. Section III describes the methodology used to find repositories and fetch their data. Section IV shows the results obtained and discusses them while Section V presents the conclusion.

II. Related Work

Cosentino, Luis and Cabot [6] presented meta-analysis of 93 research papers reporting findings from GitHub data with focus

978-1-6654-4091-2/21/$31.00 ©2021 IEEE

on their empirical methods, datasets used and limitations reported. They found that majority of studies are based on direct observation of GitHub metadata, while a little less than 25% of them apply some other methods like surveys and interviews. They discussed some common datasets used like databases and APIs Furthermore, they identified some common problems including replicability, poor sampling techniques, lack of longitudinal studies and scarce variety of methodologies.

Crowston, Wei et al. [1] reviewed the research of Free/Libre and Open Source Software development with analysis of methodology and datasets used, analysis performed, outputs and findings. They concluded that this type of development shows tremendous promise for future research and mentioned some issues including limited amount of data in certain datasets, use of possibly inaccurate self-reported data, lack of samples including unsuccessful projects and validity concerns with using different levels of data.

Kalliamvakou, Gousios, et al. [7] discussed the promises and perils of mining GitHub. They raised several concerns about repository data like the fact that large portion of repositories is small, inactive and does not provide adequate data about pull requests. They proposed avoidance strategies for each of them.

Yu, Yin, Wang et al. [8] explored different patterns of social behaviour in GitHub. They identified several interdependance and group patterns of social behaviour and examined the effect of factors like centrality in social network and following between internal and external developers on the project success. grow.

Mcdonald and Goggings [9] explored performance and participation in open source software and searched for different metrics used to measure them. They performed series of interviews with users to get their insights about their experience with GitHub and its usefulness. They also discussed the concept of distributed leadership in open source software community.

Dabbish et al. [10] conducted a series of semi-structured interviews with 24 GitHub users to find some insights about their social behaviours and collaboration. They concluded that individual goals to improve their work quality, community significance and personal relevance lead to collaboration, learning, innovation and knowledge sharing.

Tsay et al. [11] researched how different social and technical factors impact the acceptance of the contribution including following of the technical contribution norms, social connection, discussions about the contribution, submitter's status both in general community and in a project and project properties like its establishment which is inversely related to acceptance of new pull requests.

Borges et al. [12] examined different factors that impact the popularity of GitHub repositories including the programming language, their age, different complexity metrics and new features added. They also study their popularity through time to understand when and how fast do they become popular.

Varuna and Mohan [13] used time series analysis of different events and activities on GitHub including repository creation, forks, pull requests and pushes to predict the future popularity of repositories, languages and domains.

Chatziasimidis et al. [14] examined GitHub data about repositories and users to find out how different factors related to project dynamics, team size and project age and maturity affect its success and popularity.

Bana and Arora [15] searched for the most influential developers, repositories, technologies and programming languages on GitHub using social network analysis.

III. Methodology

Since GitHub [16] has become the largest code host in the world, it was chosen as a data source for the analysis. It currently hosts over 100 million repositories with more than 56 million developers. It is based on Git version control system [17] which enables tracking of all changes and activities in the project with user identities and other relevant data. Some definitions of basic git concepts are the following. [18]

Repository is the basic element of GitHub, representing the collection of all files and their revision histories related to one project. They can be divided into **original** repositories and **forks**. Fork is a personal copy of another user's repository that lives on another account where user can make changes independently.
Commit or "revision" is an individual change to a project file or set of files. It has unique identity which enables keeping record of specific changes along with who made them and when. **Merges** are commits created by bringing the contents of one branch into another. In some cases they cause conflicts which should be resolved by the user performing them. **Pull requests** are proposed changes to a repository submitted by a user which can be accepted or rejected by a repository's collaborators.

As stated in [7], most GitHub repositories have very few users, commits and low level of activity. Many of them are personal repositories or not related to complex software development. Namely, only 2.5% of projects account for 97.5% of commits in GitHub.

Furthermore, some large repositories seem big and complex when it comes to the number of commits but they are in fact forks of other large repositories with simple modifications after the fork. On the other hand, some simple repositories have high number of "artificial" commits created to test GitHub limits

In order to select several repositories for the analysis, Google BigQuery [19] was used to find several original project repositories of different sizes. 3 repositories were chosen for further analysis of their dynamics. Chosen repositories, together with their main characteristics are shown in Fig 1.

Projects	*Number of commits*	*Number of contributors*
kubernetes/kubernetes	97385	3572
pandas-dev/pandas	25928	2561
mui-org/materials-ui	15984	2330

Fig. 1. Table of chosen projects with their main characteristics

These repositories were cloned locally where they could be analysed without any network data limits. Module GitPython was used to fetch different data about commits, users and their characteristics.

One of the main information related to commit is the complexity of the change made by the commit which can be estimated using following metrics explained by Choudhary et al. [20]:

- number of lines inserted: The more complex the change is, the more lines of code will probably have to be added to achieve it.
- number of lines deleted: It can be used to estimate how much the change interferes with an existing code.
- number of total line changes: number of lines inserted + number of lines deleted
- time passed since parent commit: Change complexity is an important factor when it comes to time needed to perform it, however other factors like developer's availability can cause drastic changes in this parameter so it should not be used as the main metric.

Besides commits, authors (or users) were examined by their activity and properties of their commits. Following author data was observed:

- number of lines changed
- total number of line changes
- average number of line changes by commit
- total time between all user commits and their parent commit
- total number of all files changed by user in all commits
- number of distinct files changed by user in all commits
- number of merge commits performed by user

IV. Results and Discussion

In following figures repositories from Fig. 1. were shown in blue, orange and green colour, consecutively.

Using several parameters observed, many rules and patterns can be found.

Fig. 2. – Fig. 6 are related to commit data while Fig. 7 – Fig. 10 are related to user data

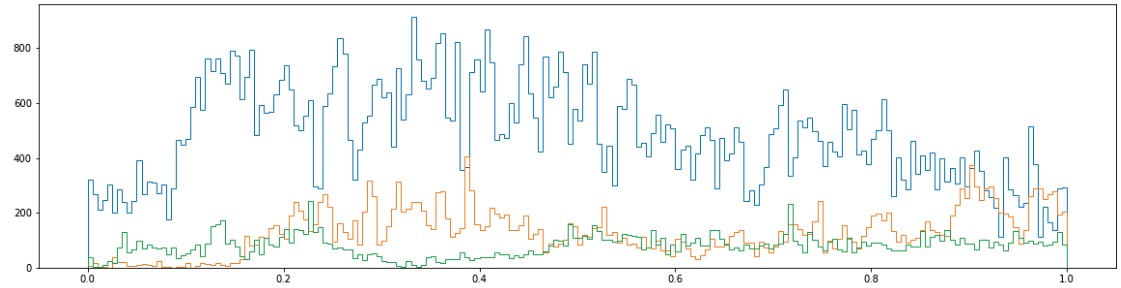

Fig. 2. Histogram of commits over time

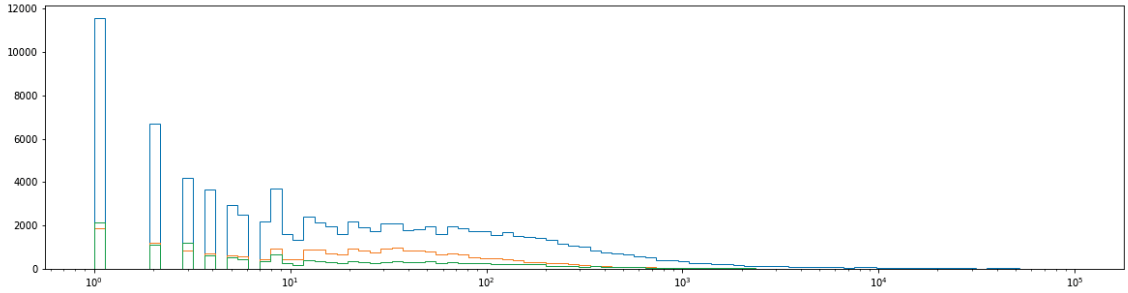

Fig. 3. Histogram of commits by number of lines inserted (logarithmic scale on x axis)

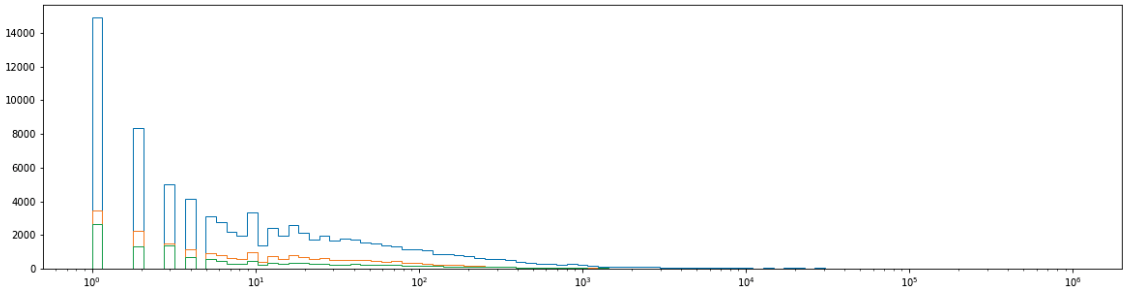

Fig. 4. Histogram of commits by number of lines deleted (logarithmic scale on x axis)

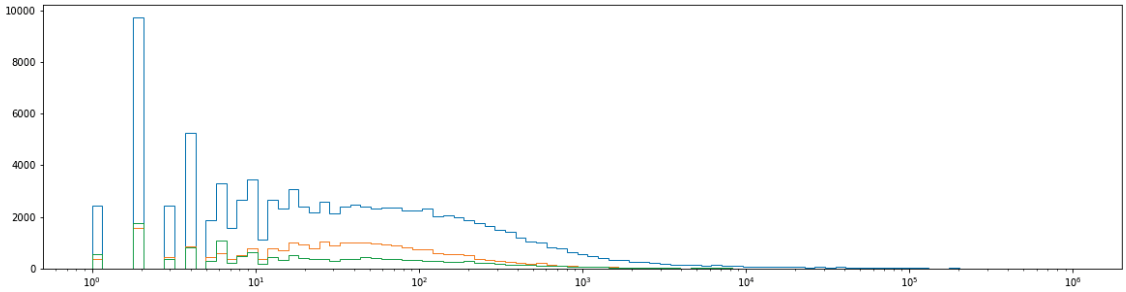

Fig. 5. Histogram of commits by total number of line changes (logarithmic scale on x axis)

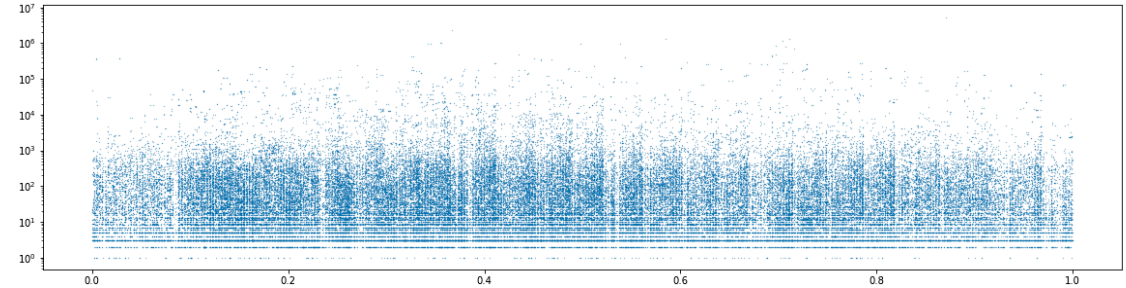

Fig. 6. All commits in repository kubernetes/kubernetes shown by time on x axis and number of line changes on y axis in logarithmic scale

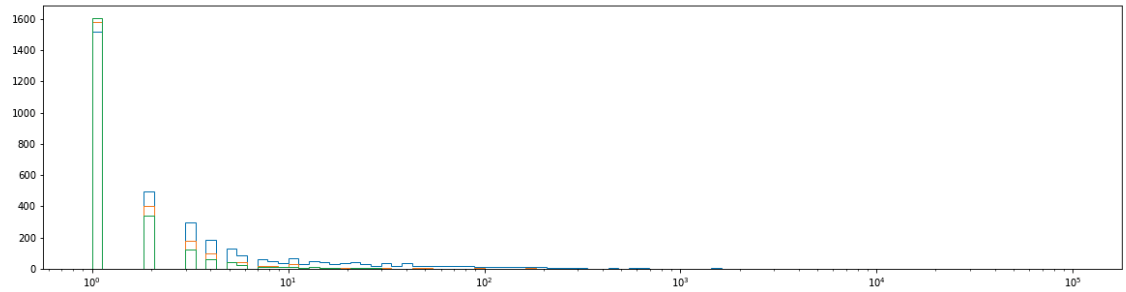

Fig. 7. Histogram of users by number of commits (logarithmic scale on x axis)

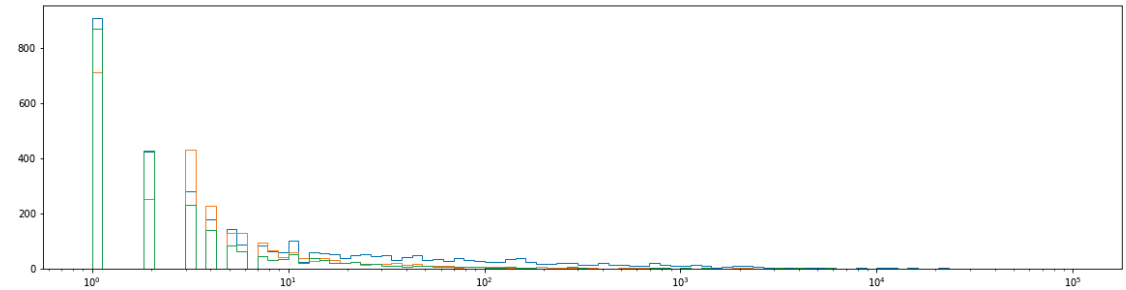

Fig. 8. Histogram of users by number of files changed (logarithmic scale on x axis)

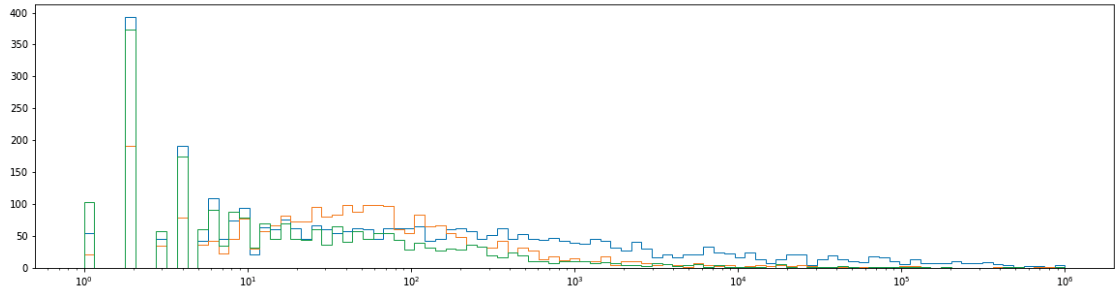

Fig. 9. Histogram of users by number of line changes (logarithmic scale on x axis)

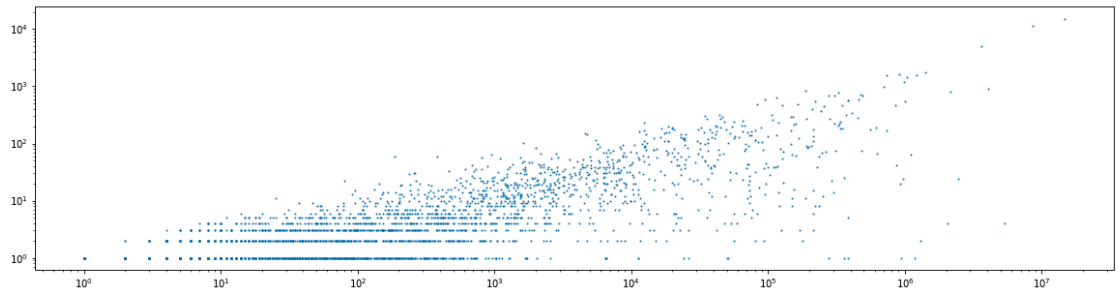

Fig. 10. All users in repository kubernetes/kubernetes shown by total number of line changes on x axis and number of commits on y axis.

Majority of contributors had very little activity in terms of commits (Fig. 7.), line (Fig. 8.) and file (Fig. 9.) metrics. Little contributors are often users of the product that want to fix some specific error or change specific feature in the project. Large contributors usually make larger changes, often in the project core. Also, the majority of commits have a very low number of line changes (Fig. 3. – Fig. 5.), for example commits with 1 line insertion and 1 deletion or 2 line insertions and 2 deletions were very common.

User activity shows some seasonal properties (Fig. 6.) related to specific parts of days, weeks and years. Some periods of reduced activity occur during nights, weekends and holidays and periods of higher activity usually come immediately before them, as contributors try to publish their changes before periods of lower activity.

Besides periodical changes, contribution activity changes as a consequence of product popularity, project establishment and needs for new features. Fig. 2. shows that project pandas had low level of activity for a long time until it became more popular. Project kubernetes/kubernetes shows high level of activity since the beginning but seems to slightly decrease in activity over time. Project mui-org/materials-ui has fairly stable level of activity except for one period when it was much lower.

Fig. 10. shows some differences in committing habits of different users. Users that made the most line changes also have high number of commits but some users made big changes that are divided into small number of large commits.

Obtaining GitHub data is commonly discussed problem, different APIs and databases are used for that problem but most of them contain some limits regarding the number of queries or the amount of data they can process within a certain time. It is important to choose the tool that offer the data needed in a satisfactory manner.

Some problems with data about commits include their logical connection since two consecutive commits can be logically unrelated. Some heuristics dealing with that problem include analysis of changes of the same functions, files or other entities of code.

Commit does not always represent one logical task since users have different committing habits, as shown in Fig. 10. Some commits are very big, consisting of many different tasks and issues resolved at once and breaking them into smaller logical parts could be very challenging task.

Some git commands like “git rebase” can alter the repository history, hide some information about branches and merges and cause other irregularities like causing the situation that the parent commit is later in time than the child commit. One of the simplest solutions to this problem is to assign some constant time difference in situations where two consecutive commits are in the wrong time order.

V. Conclusion

In this work, the concept of open source software development was briefly explained with its peculiarities and other related paradigms. Open source software environment GitHub was used to collect data and draw conclusions about project dynamics. Common properties observed were big part of commits with very small changes and users with very low level of activity, periodical behaviours. Major differences were found

in project dynamics over time depending on popularity and several other factors.

This work has many limitations and a big room for further development. GitHub contains a lot of other data that can be exploited and analysed like data about repository popularity, forks, pull requests, following between users etc. These data can greatly improve the understanding of the dynamics and its social properties.

Commit complexity measures used are simple to obtain but often do not give good estimation. Lines of code vary in complexity and changes in them can be smaller or bigger. Time measures are affected by other factors like user availability. Many advanced measures of commit complexity can be applied regarding files, objects, functions, methods, lines of code and other entities. They may give more accurate estimate but finding them may slow up the process or limit it to smaller set of programming languages and technologies. Also, the information about the time needed can possibly be improved by taking into account seasonal properties of that time.

Future work on this topic may include:

- improving the analysis by using more repositories, their data and complex metrics to improve understanding of different factors in open source software development
- application of machine learning techniques to make some predictions and estimations about the projects, their success, type, popularity, future changes etc.
- development of a model to simulate dynamics of open source software development projects to predict the outcomes and explain how different individual, project and social factors affects its' dynamics and success. One possible approach is to use agent-based models [21] which consist of autonomous objects (agents) with rules and relationships that control their behaviour. Bankes [22] mentions three reasons for the importance of agent-based models to social science including the unsuitability of competing modelling formalisms to address the problems of social science which suggests that agent-based modelling may be a good technique to approach this problem.

ACKNOWLEDGEMENT

This work was funded by Ministry of Science, Education and Sports of the Republic of Croatia and Croatian Science Foundation project TAIDE (Team Adaptability for Innovative Product Development, grant number 7269, www.taide.org)

REFERENCES

[1] Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012, February). Free/Libre Open-Source Software development: What we know and what we do not know. *ACM Computing Surveys*, Vol. 44.

[2] Chesbrough, H. (2012, July). Open innovation: Where we've been and where we're going. *Research Technology Management*, Vol. 55, pp. 20–27. https://doi.org/10.5437/08956308X5504085

[3] Walli, S., Gynn, D., & Von Rotz, B. (2005). *The Growth of Open Source Software in Organizations*.

[4] Chesbrough, H. W., Vanhaverbeke, W., & West, J. (2008). *Open Innovation: Researching A New Paradigm*.

[5] West, J., & Gallagher, S. (2006). Challenges of open innovation: the paradox of firm investment in open-source software. *R and D Management*, *36*(3), 319–331. https://doi.org/10.1111/j.1467-9310.2006.00436.x

[6] Cosentino, V., Luis, J., Izquierdo, C., & Cabot, J. (2016). Findings from GitHub: Methods, Datasets and Limitations. *Proceedings of the 13th International Conference on Mining Software Repositories*.

[7] Kalliamvakou, E., Gousios, G., Kelly Blincoe, T., Singer, L., German, D. M., & Damian, D. (n.d.). The Promises and Perils of Mining GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*.

[8] Yu, Y., Yin, G., Wang, H., & Wang, T. (2014). Exploring the Patterns of Social Behavior in GitHub. *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies - CrowdSoft 2014*.

[9] Mcdonald, N., & Goggins, S. (n.d.). Performance and Participation in Open Source Software on GitHub Work-in-Progress: CSCW CHI 2013: Changing Perspectives. *CHI '13 Extended Abstracts on Human Factors in Computing Systems on - CHI EA '13*. New York, New York, USA: ACM Press.

[10] Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository Human Factors; Design. *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work - CSCW '12*.

[11] Tsay, J., Dabbish, L., & Herbsleb, J. (2014). Influence of Social and Technical Factors for Evaluating Contribution in GitHub. *Proceedings of the 36th International Conference on Software Engineering*.

[12] Borges, H., Hora, A., & Valente, M. T. (2017). Understanding the factors that impact the popularity of GitHub repositories. *Proceedings - 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016*, 334–344.

[13] Varuna, T. V., & Mohan, A. (2019). Trend Prediction of GitHub using Time Series Analysis. *2019 10th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2019*.

[14] Chatziasimidis, F., & Stamelos, I. (2016). Data collection and analysis of GitHub repositories and users. *IISA 2015 - 6th International Conference on Information, Intelligence, Systems and Applications*.

[15] Bana, R., & Arora, A. (2018). Influence Indexing of Developers, Repositories, Technologies and Programming Languages on Social Coding Community GitHub. *2018 11th International Conference on Contemporary Computing, IC3 2018*

[16] GitHub, Retrieved Jan 30, 2021 from https://github.com

[17] Git source control management, Retrieved Jan 30, 2021 from https://git-scm.com

[18] A Git Glossary, Retrieved Jan 30, 2021 from https://git-scm.com/docs/gitglossary

[19] Google BigQuery, Retrieved Jan 30, 2021 from https://cloud.google.com/bigquery

[20] Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A., & Catal, C. (2018). Empirical analysis of change metrics for software fault prediction. *Computers and Electrical Engineering*, *67*, 15–24. https://doi.org/10.1016/j.compeleceng.2018.02.043

[21] Macal, C. M. (2016). Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, *10*(2), 144–156. https://doi.org/10.1057/jos.2016.7

[22] Gilbert, N., & Bankes, S. (2002, May 14). Platforms and methods for agent-based modeling. *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 99, pp. 7197–7198. https://doi.org/10.1073/pnas.072079499