[1] Ulvi Shakikhanli*
[2] Vilmos Bilicki

Analyzing Branching Strategies for Project Productivity: Identifying the Preferred Approach

Journal of Electrical Systems

Abstract: - The productivity of software development processes has been the subject of extensive research over the past few decades. Given the broad scope of the topic, even establishing a definitive definition for productivity has posed a challenge. This paper aims to investigate the influence of both repository structure and branching strategy on software productivity, a relatively unexplored area of study. As the choice of branching strategy is made early in the development process, its impact can be substantial on overall productivity. The findings of this study reveal that high productive projects tend to favour Multi repository structures over Mono repository ones. Moreover, highly productive projects predominantly adopt branching strategies such as Github Flow and GitFlow, as opposed to the Trunk-based approach. The analysis also encompasses various metrics including commit count, branch count, and programming languages utilized in the development process. With the help of those analysis a new approach for the calculation of productivity level and the estimation of development period have been proposed. By utilizing this robust dataset, this study provides objective insights into the impact of repository structure and branching strategy on software productivity.

Keywords: Data Mining, Github Mining, Software development productivity, Repository Structure, Branching Strategy.

I. INTRODUCTION

This research explores software development productivity by examining various aspects, including repository structure, and branching strategies. The upcoming chapter will discuss different approaches to measure productivity and focus on three primary branching strategies in modern Distributed Version Control Systems (DVCs): Trunk-based, GitFlow, and GitHub Flow [1].

The Trunk-Based strategy, popular on GitHub, is noted for its simplicity and efficiency. It relies on a single deployable master branch, simplifying the development process by avoiding multiple branches. In contrast, the GitFlow strategy is more intricate, involving multiple branches like Master, Develop, Feature, and Hotfix for organized development and effective collaboration.

GitHub Flow, a simpler variation of GitFlow, primarily uses a Master branch and feature branches, facilitating modular and iterative development. Features are developed independently and merged back into the Master branch through pull requests.

The paper aims to clarify its findings by addressing four key questions, which will succinctly convey the primary outcomes of the research.

RQ 1: Which branching strategy is more preferred by high and low productive projects?

RQ 2: Is there any relation between development period and team size with productivity of projects?

II. RELATED WORK

The project productivity was one of the most important research topics for several decades and it is still a highly debated topic among both academic and industrial communities. Each paper in this topic tries to give its own definition for the terms of "project productivity" and measures it according to the different parameters or characteristics of the project. For example, paper [2] describes productivity as a ratio of product size to project effort:

$$Productivity = Size / Effort \quad (1)$$

However, this method obviously cannot be implemented to all types of projects when there is no connection between the size of the project and the effort spent to develop it. Such scenarios can be seen especially in web projects. Also, this type of productivity can force developers to create a huge size of projects which have no real value. Using this approach paper [3] improves it and creates a new formula for the calculation of productivity:

$$Productivity = AdjustedSize / Effort \quad (2)$$

In the formula (2) AdjustedSize is includes only those size measures that together have a significant relationship with effort. This approach, basically similar to the previous one, measures productivity according to the effort spent.

[1] Doctoral School of Computer Science, Faculty of Science and Informatics, University of Szeged, H-6720, Szeged, Hungary. ulvi@inf.u-szeged.hu, ORCID ID: 0000-0002-6947-3119
[2] Doctoral School of Computer Science, Faculty of Science and Informatics, University of Szeged, H-6720, Szeged, Hungary. bilickiv@inf.u-szeged.hu, ORCID ID: 0000-0002-7793-2661
* Corresponding Author Email: ulvi@inf.u-szeged.hu

However, they add some exceptions like maintenance jobs and other work which is not included in this concept. But this approach was created in 2004 and the software and web development environment have changed a lot since back then. That's why this approach cannot be accepted as fully accurate nowadays.
Quality: They use LGTM [4] for measuring quality of code. Applying machine learning makes this approach a more reliable approach than others but using only commits as measurement can be accepted as its main weakness. Here we have to mention that using commits for productivity measurement is a pretty popular choice among most of the papers which we have seen so far. For example, papers [5], [6] and [7] use the total amount of commits as a project's productivity.
Paper [7] also tries to show the role of branching strategies in productivity. Analysis done on nearly 3000 projects but none of the branching strategies have been considered. Instead of that they checked the overall properties of the branching strategy.

III. METHODOLOGY

3.1. Data

The database [8] comprises a comprehensive collection of over 50,000 projects, encompassing both Mono repository and Multi repository structures. Among these projects, there are 16,958 Multi repositories and 33,594 Mono repository projects. Each project is stored as a JSON format file, containing vital information about the project and its corresponding repository.

3.2. Productivity

Numerous methodologies for measuring productivity have been extensively discussed in the preceding chapter. Notably, a majority of these methodologies rely on a limited set of project parameters to determine productivity. In this study, we adopt the approach employed in [9]. There are several compelling reasons for this choice:

- The selected approach leverages a more comprehensive set of project details, offering a distinct advantage in terms of accuracy and reliability compared to alternative methodologies.
- The researchers in [9] specifically utilized open-source projects from the GitHub platform, aligning it well with the nature of our database.

In [9], the authors employ three distinct models to identify active bursts, namely Maximal Sum Segments [10], Kleinberg Burst Detection [11], and Hidden Markov Model [12].
Before checking their approach authors used additional evaluation metrics in order to be able to compare all three approaches. The authors aim to evaluate the bursts generated by the aforementioned models by comparing them to a lexically coherent segmentation of the project timeline using Beeferman et al. [13]'s Pk as the metric of comparison. As it is written in the paper [9] an alternative method involves analysing the communication among project developers and identifying consecutive days with similar conversation topics as "lexically coherent bursts". These bursts of activity, which closely resemble the identified lexical segments, not only indicate high levels of activity but also suggest work towards similar objectives, thereby representing a cohesive unit of work. The model of lexical cohesion that we utilize is a widely recognized text segmentation technique called Text Tiling [14].

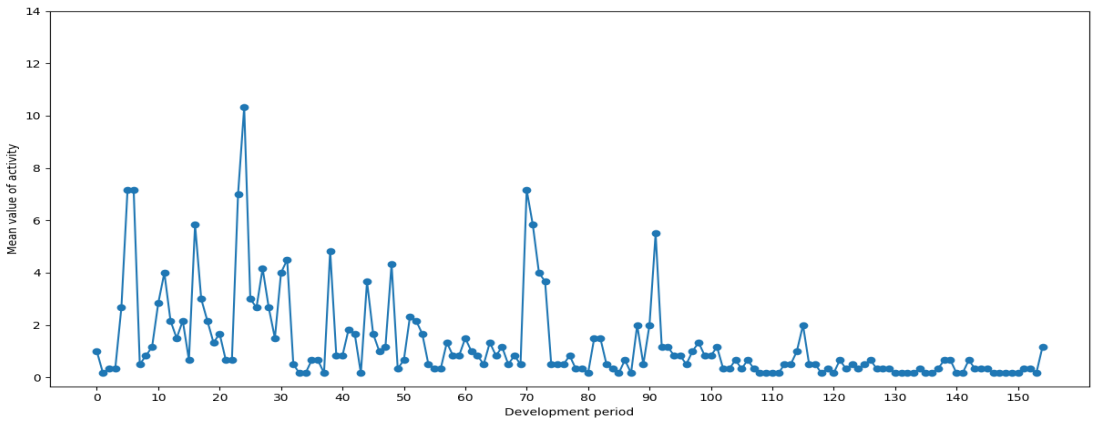

Figure 3.2.1 Burst stream of Mono repository project.

States exhibiting activity values below a specified threshold were categorized as "low active," while those surpassing the threshold were classified as "active." For each project, the days assigned the "active" state were recorded. Subsequently, a consecutive sequence of these "active days" was grouped together to form a burst, with a maximum gap of 3 days allowed between each subsequent active day. By creating bursts in this manner, the project's activity timeline is divided into distinct segments characterized by bursts of activity, facilitating meaningful comparisons with burst segmentations.

IV. RESULTS

We present a detailed overview of the statistical results obtained from our analysis. These findings offer compelling evidence regarding the relationship between burst lengths and productivity states, enabling us to draw meaningful conclusions about the efficiency and effectiveness of different projects. Furthermore, our measurements provide valuable insights into the role of various project parameters in shaping productivity outcomes.

4.1 Usage percentage of three main branching strategies in different categories of productivity levels.

The choice of branching strategy in software development projects has a significant impact on their productivity. In this section, we examine the usage percentages of the three main branching strategies, namely Trunk-based, Github Flow, and GitFlow, across different categories of productivity levels in Mono and Multi Repository projects. The following statistics, derived from Table 4.1, shed light on the relationship between the repository structure and the productivity of the projects.

Table 4.1 Percentage share of 3 main branching strategies in Mono and Multi Repository projects

	High Productive	**Low Productive**	**Non-Productive**
Mono Repository	Trunk-Based: 33.2 % Github Flow: 45.8 % GitFlow: 21.0 %	Trunk-Based: 61.4 % Github Flow: 30.3 % GitFlow: 8.2 %	Trunk-Based: 85.2 % Github Flow: 12.0 % GitFlow: 2.7 %
Multi Repository	Trunk-Based: 16.1 % Github Flow: 45.3 % GitFlow: 38.6 %	Trunk-Based: 44.2 % Github Flow: 36.6 % GitFlow: 19.3 %	Trunk-Based: 73.8 % Github Flow: 17.1 % GitFlow: 9.1 %

In Mono Repository projects, Table 4.1 reveals distinct preferences in branching strategies across different productivity levels. High productive projects predominantly adopt the Github Flow strategy (45.8%), followed by Trunk-based (32.3%) and GitFlow (21%) strategies. In contrast, low productive projects favour the Trunk-based approach (61.4%), with Github Flow (30.3%) and GitFlow (8.2%) being less prevalent. Non-productive projects overwhelmingly prefer the Trunk-based strategy (85.2%), with minimal use of Github Flow (12.0%) and GitFlow (2.7%), highlighting a trend towards straightforward, agile methods in less productive contexts.
In Multi Repository projects, Table 4.1 illustrates varied preferences in branching strategies across productivity levels. High productive projects mainly use Github Flow (45.3%), followed by GitFlow (38.6%) and Trunk-based (16.1%) strategies. Low productive projects show a preference for Trunk-based (44.2%), Github Flow (36.6%), and GitFlow (19.3%). In nonproductive projects, Trunk-based strategy is most prevalent (73.8%), with lesser use of Github Flow (17.1%) and minimal adoption of GitFlow (9.1%), indicating a trend towards more straightforward, agile approaches in less productive scenarios.

4.2 Comparison of development period and team size in different branching strategies according to the productivity level.

As we have discussed commit count to see the workload in different branching strategies and productivity levels there is also needed to see two major parameters: development period and team size. These factors are highly important during the planning phase of software development and can help developers to save huge amounts of time and effort and work more productively.

4.2.1 Mono Repository

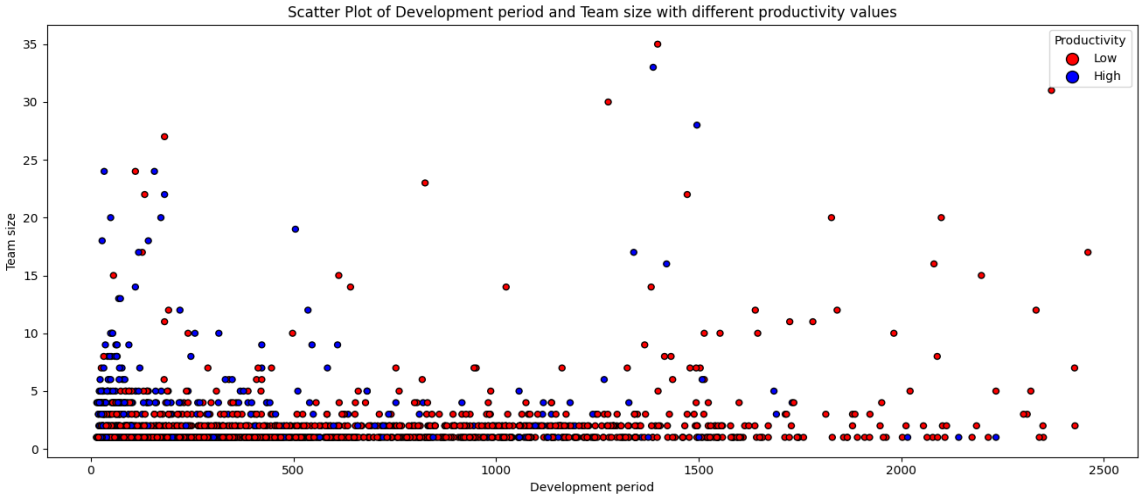

Figure 4.1 Scatter Plot of Development period and Team size with different productivity values in Mono repository projects.

Figure 4.1 shows that most of the highly productive projects have a development period lower than 1000 days which is approximately 3 years, and the most popular team size is between 3-20. Of course, the team size and development period are dependent on several factors which are not possible to measure. But this real-world data still gives us important results which have not been observed before.

4.2.2 Multi Repository

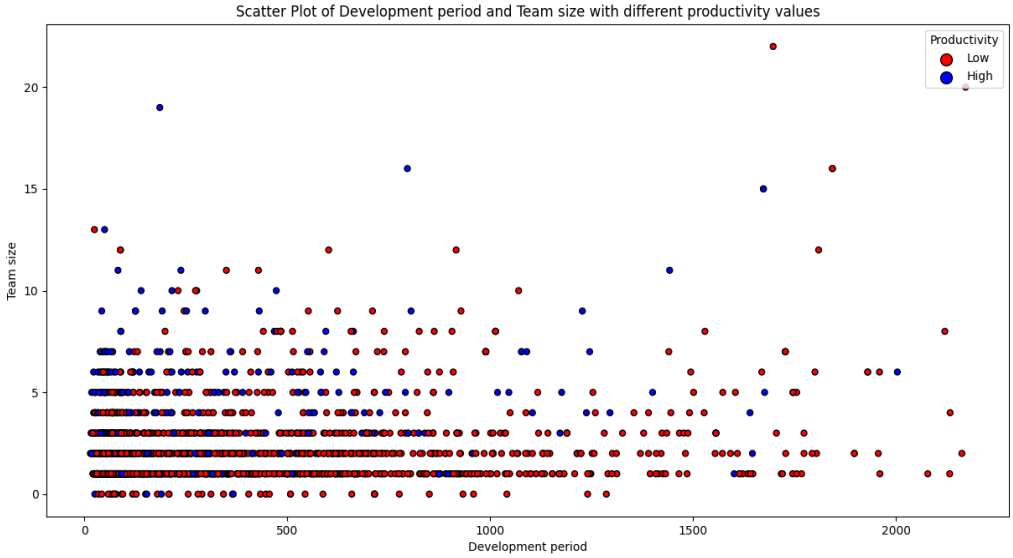

Figure 4.2 Connection Development period and Team size with different productivity values for Multi repository projects.

Figure 4.2 shows that similar results also can be observed in Multi repository projects. In this case also high productive projects have much shorter development periods than low productive ones. It lets us assume that both Mono and Multi repository projects have similar preferences about development period and team size in order to be more productive.

V. Model Training

5.1 Machine Learning Model for Productivity level.

According to the found relationships between different parameters of repository and its productivity it is possible to propose new method for the calculation of productivity level. There are four main types of features in our database:

- *String format*: Repository name, repository types, branching strategy and so on.
- *Text format*: Description of repository, commit comments and so on.
- *Integer format*: Commit count, branch count, developer count and so on.
- *Date format*: Creation date of repository, commit date and so on.

Not all of these features are suitable for our model. For example, the features like “contributors”, “branches”, “pull requests”, “issues”, “issue comments”, “pull request comments”, “events” and “commits”. All of these features contain a list of dictionaries with several values inside like content of features, data of added content and so on. In most of the cases only the count of these features has been used. This approach has been chosen in order to simplify the feature generation process.

Table 4.1 Accuracy results of model training process.

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.9003	0.8125	0.9019	0.8533
Decision Tree	0.6802	0.4952	0.6705	0.5765
Random Forest	0.9344	0.8410	0.9274	0.8948
Support Vector Machine	0.6397	0.4732	0.6221	0.5255

As it can be seen from the *Table 4.1* trained models gives us high accuracy results with *Random Forest* being the most suitable in our case. This approach has several advantages over the previous case. First of all, since it is using the parameters like branching strategy, team size, development period and so on this model can be implemented almost all type of project and repositories. Secondly after creating this model researchers can easily use, it for other projects as well.

5.2 Machine Learning Model for Development period.

Development period of the project can be one of the essential parameters during both planning and development phase. After analysing the connection between development period and other parameters of projects it become evident that a ML model can be used for the estimation. Our approach here is much more similar with the previous case.

As it was in previous case about the prediction of productivity in this approach it is also very important to choose the correct set of features for the best result. Some of the features which have been mentioned in previous cases contain much more complex structure than needed for our model. That's why those features have been modified. The dataset provided information on several aspects of the GitHub projects. The features used for the model were selected based on their potential relevance to the development period. These features included:

- Number of Commits
- Number of Contributors
- Number of Branches
- Number of Pull Requests
- Number of Issues
- Productivity (High, Low, None)
- Branching Strategy (Trunk-Based, GitFlow, Github Flow)

The machine learning model employed for this task was the Random Forest Regressor. This model was chosen for its robustness to overfitting and its ability to handle a wide range of data types and complex relationships within the data.

The target variable, initially provided in days, was converted to represent the development period in months. This conversion was done to align the predictions more closely with typical project planning timelines, which are often measured in months. Furthermore, to simplify the interpretation of the results, the development periods were rounded to whole months for the final predictions.

The model's performance was evaluated using three key metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and the coefficient of determination (R-squared). The results were as follows:

- Mean Absolute Error (MAE): 3.40 months.
- Mean Squared Error (MSE): 52.42 months^2
- R-squared (R^2): 0.441

These results indicated a low level of error, with the model predicting the development period, on average, with an error of approximately 3.40 months. The R^2 value was medium, indicating that the model significantly explains the variability in the development periods of the projects.

VI. 6. DISCUSSION

RQ 1. Which branching strategy is more preferred by high and low productive projects?

Results of low productive and high productive projects shows that Trunk-based approach has nearly 30% value while Github Flow and GitFlow have values 46% and 21% respectively. It means that the Trunk-based approach still has its place among high productive projects, but it is not the same amount as Github Flow one. Github Flow has 30% in low and 46% in high productive projects which show first of all it is significant popularity among real world projects and also being the most preferred branching strategy for high productive projects. The value of GitFlow is much smaller even than Trunk-based strategy but there is little information which has to be considered. The prevalence of Trunk-based projects in databases outweighs the number of GitFlow projects, indicating that a substantial proportion of projects utilizing GitFlow are either highly productive or low in productivity. This observation can be attributed to the intricate nature of the GitFlow branching strategy, which necessitates its implementation primarily in professional and mission-critical projects.

RQ 2: Is there any relation between development period and team size with productivity of projects?

The relationship between team size, programming language, and productivity in different repository structures is understudied, as noted in Paper [9]. Existing research often focuses on Agile methodologies or team motivation, overlooking team size and development period. This study examines their correlation with project productivity.

Mono Repository:

Figure 4.1's scatter plot shows the relation between team size and development period in Mono repository projects. Combining results for Github Flow and GitFlow, it reveals that high productive projects typically have development periods under 2 years and team sizes below 15, a trend consistent in low productive projects. These projects, being open source, commonly have smaller teams. Projects exceeding 2 years in development are often less productive.

Multi Repository:

Contrasting with Mono repositories, Figure 4.2 shows Multi repository projects generally have longer development periods, as indicated by the scatter plot's wide dispersion of high productive project points. While team sizes are similar to Mono repositories, high productive Multi repository projects are marked by notably longer development periods.

VII. Conclusion.

The paper delves into the impact of branching strategies in software development, particularly in relation to project productivity within Mono and Multi repository structures. It reveals that most non-productive projects predominantly use the Trunk-based branching strategy, despite its limitations, particularly in handling new features and code issues. In contrast, more productive projects demonstrate a diverse use of strategies, with Github Flow (46%) and GitFlow (21%) being more prevalent than the Trunk-based approach. This diversity underscores the effectiveness of Github Flow in high-productivity scenarios, while the complex GitFlow strategy, though less common, is significant in professional and mission-critical projects. Additionally, the paper explores the relationship between team size and development period in these projects. It finds that for Mono repository projects, most high-productivity cases have development periods under 2 years with team sizes below 15, a pattern attributed to the nature of open-source projects. In contrast, Multi repository projects often have longer development periods, suggesting that despite similar team sizes, these projects require more time, likely due to their increased complexity. This comprehensive analysis offers valuable insights into the dynamics of branching strategies and team composition in relation to project productivity.

Financing

The authors did not receive financing for the development of this research.

Conflict Of Interest

The authors declare that there is no conflict of interest.

Authorship Contribution:

1. Conceptualization: Ulvi Shakikhanli, Vilmos Bilicki
2. Data curation: Ulvi Shakikhanli
3. Formal analysis: Ulvi Shakikhanli
4. Acquisition of funds: Vilmos Bilicki
5. Research: Ulvi Shakikhanli, Vilmos Bilicki
6. Methodology: Ulvi Shakikhanli
7. Project management: Vilmos Bilicki, Ulvi Shakikhanli
8. Resources: Ulvi Shakikhanli
9. Software: Ulvi Shakikhanli
10. Supervision: Vilmos Bilicki
11. Validation: Vilmos Bilicki
12. Display: Ulvi Shakikhanli
13. Drafting - original draft: Ulvi Shakikhanli
14. Writing - proofreading and editing: Ulvi Shakikhanli

References

[1] GitKraken, “What is the best Git branch strategy?”, https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy, Available 25/08/2023.

[2] A. MacCormack, C. Kemerer, M. Cusumano, and B. Crandall, “Trade-Offs between Productivity and Quality in Selecting Software Development Practices,” IEEE Software, pp. 78-79, Sept./Oct. 2003.

[3] Kitchenham, Barbara, and Emilia Mendes. "Software productivity measurement using multiple size measures." IEEE Transactions on Software Engineering 30.12 (2004): 1023-1035.

[4] LooksGTM, https://github.blog/2022-08-15-the-next-step-for-lgtm-com-github-code-scanning/, Available 25/08/2023.

[5] J. Gamalielsson and B. Lundell. Long-term sustainability of open source software communities beyond a fork: A case study of libreoffice. In IFIP International Conference on Open Source Systems, pages 29–47. Springer, 2012.

[6] B. Vasilescu, D. Posnett, B. Ray, M. G. van den Brand, A. Serebrenik, P. Devanbu, and V. Filkov. Gender and tenure diversity in github teams. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, pages 3789–3798. ACM, 2015.

[7] Zou, Weiqin, et al. "Branch use in practice: A large-scale empirical study of 2,923 projects on github." 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2019.

[8] Ulvi Shakikhanli and Vilmos Bilicki. "Comparison between mono and multi repository structures." Pollack Periodica (2022).

[9] Choudhary, Samridhi Shree, et al. "Modeling coordination and productivity in open-source GitHub projects." School of Computer Science, Carnegie Mellon University (2018).

[10] Walter L Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In ISMB, volume 99, pages 234–241, 1999.

[11] Jon Kleinberg. Bursty and hierarchical structure in streams. Data Mining and Knowledge Discovery, 7(4):373–397, 2003.

[12] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2):257–286, 1989.

[13] Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. Machine learning, 34(1):177–210, 1999.

[14] Marti A Hearst. Multi-paragraph segmentation of expository text. In Proceedings of the 32nd annual meeting on Association for Computational Linguistics, pages 9–16. Association for Computational Linguistics, 1994.