Finding your digital sibling

Grouping GitHub projects that share certain attributes based on interactions and activities

Rowan de Bruin

Supervisor(s): Sebastian Proksch, Shujun Huang

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 28, 2024

Name of the student: Rowan de Bruin
Final project course: CSE3000 Research Project
Thesis committee: Sebastian Proksch, Shujun Huang , Julia Olkhovskaia

Abstract

This study explores the feasibility of categorizing GitHub projects based on their interactions and activities, aiming to assist both researchers and practitioners in navigating the vast landscape of open-source software. Through experiments and analysis, key attributes contributing to project categorization are identified, paving the way for effective grouping of projects in terms of interactions and activities.

Findings indicate distinct clusters among GitHub projects, highlighting the influence of interactions and activities on project categorization. The study underscores the importance of refining grouping algorithms and improving project categorization methods for future research.

Future work could involve developing user-friendly tools to facilitate project discovery and exploring correlations between interaction-related metrics and project development dynamics. Overall, this study contributes to advancing our understanding of project categorization on GitHub, facilitating more efficient knowledge sharing and collaboration within professional fields.

1 Introduction

Open-source software is vast and diverse, with platforms such as GitHub providing access to millions of projects for both researchers and practitioners. Despite the abundance of resources, newly initiated projects frequently encounter challenges in identifying suitable role models. It would prove highly advantageous for struggling projects to identify other projects with similar properties. However, the sheer volume of projects renders navigation almost impractical.

The subsequent sections delve into the related work, methodology, experiments, and results. The main research question we want to answer is: "Can we group GitHub projects based on interactions and activities?" With research question 1 being: "which attributes that are related to interactions and activities would be suitable when comparing two open-source projects?" And research question 2 being: "Does grouping GitHub repositories based on inter-action and activities bring the same or different results as compared to grouping on different metrics?"

The related work highlights existing algorithms such as RepoPal [10] and CLAN [9] for grouping projects based on similarities and introduces the Reaper framework by Kalliamvakou et al [7]. for identifying engineered software projects in GitHub repositories. The proposed methodology involves building an algorithm to measure project similarity based on interaction and activities. The first research question focusses on finding the right metrics to use for this algorithm. The second research question, conducted in two parts, aims to compare results of the algorithm to another algorithm based on different metrics. The results indicate successful grouping of projects and highlight differences when compared to other metrics, emphasizing the multidimensional nature of GitHub repositories.

The discussion section interprets the findings, addressing the main research questions and acknowledging limitations. Research question 1 focusses on finding the best metrics for grouping projects based on interactions and activities. Research question 2 compared results with the CrossSim tool [3], revealing differences due to variations in metrics. The conclusion emphasizes the feasibility of automated grouping, providing valuable insights for future research and practical applications in project management. The paper concludes with a section on responsible research, emphasizing ethical considerations and commitment to user privacy and data security.

The references include key works in the field, such as the research by Kalliamvakou et al [7]. on mining GitHub, the CrossSim tool [3], and studies on detecting similar repositories. The future work section suggests the development of a comprehensive software tool based on the proposed algorithm, highlighting its potential importance for businesses.

In summary, this paper addresses the pressing issue of identifying similarity accross GitHub projects focusing on interactions and activities, contributing valuable insights for both researchers and practitioners in the realm of open source software development.

2 Related work

There are a couple of others who have attempted to automate grouping of projects. One of these algorithms is called RepoPal. RepoPal is a tool that aims to identify multiple GitHub projects that share similarities. It achieves this through three heuristics: Readme files, repository users' interests, and repositories of the same users in a short amount of time [10]

Another tool, CLAN, which stands for "CLosely relAted applicatioNs," also attempts to identify similarity between multiple applications. However, CLAN does this by looking at different characteristics, namely packages and class hierarchies [9].

Furthermore, you also have an interesting piece of research with the challenge of discerning genuine projects from noise within the extensive GitHub repository landscape, made by Kalliamvakou et al. [7]. They propose "Reaper," a framework designed to identify repositories demonstrating clear evidence of engineered software projects. This is achieved through the measurement of specific software engineering practices. Their evaluation on a dataset of 1,857,423 GitHub repositories reveals that their classifiers surpass traditional metrics like GitHub Stargazers. The classifiers achieve a balanced precision (82%) and recall (86%). These findings highlight the significance of advanced frameworks such as Reaper in bolstering the reliability of research outcomes in software engineering.

Another interesting piece is a paper from Jailton Coelho et al. [2]. In this paper, the authors address the growing concern of sustainability in open-source software projects managed by a limited number of volunteers. They propose a machine learning approach to identify GitHub projects lacking active

maintenance, aiming to alert users and potentially encourage developers to take over. The method, validated with real open source developers, demonstrates a precision of 80% and a recall of 96%. The proposed model offers a valuable tool for users and developers to identify projects at risk of becoming unmaintained.

3 Methodology and Experiment Setup

With our experiments we want to answer two questions:

1. Which attributes that are related to interactions and activities would be suitable when comparing two open-source projects?
2. Does grouping GitHub repositories based on inter-action and activities bring the same or different results as compared to grouping on different metrics?

To answer these questions, we first design an algorithm which can give the similarity between multiple GitHub projects, and then, based on this similarity, group these projects.

3.1 Building the algorithm

The first part of the algorithm we need, is the functionality to read a list of repository names. We start by injecting a ".txt" file with the repository names. Now to read this file by using a FileReader, reading the file line by line and putting these names into a list. Next up, we need a method to pull these repository from GitHub, so that we can analyse them. This is made possible by the GitHub API. But since the GitHub API does not support everything we need, we can also use a wrapper around this API to extract all the information we need. The wrapper we use is made by Kohsuke Kawaguchi [1].

But because we noticed that grabbing all the repositories from GitHub might take a lot of time, we needed to speed this process up. We are doing this by saving all our intermediate results to file, so that the algorithm doesn't need to fetch all the repositories and files each time we run it.

Next up, we need to write the methods that can analyse the repositories. We have decided to write this code in Java, since at the end of this project this project will we combined with 4 other projects. And since Java was the most common ground for us as a programming language, we chose this.

Continuing on the methods, we make a method for each metric that we want to analyse. We then run these methods and check if this corresponds to our intermediate results on file, and change/add to the file where needed.

We have now made the functionality to analyse the repositories. But before we can actually compare the similarity between these projects, we need to first define when a project is similar. For this, we calculate a similarity score.

Now that the algorithm can give us the similarity between the different repository's, there is still one thing left to do, grouping them. We do this in the following way:

1. Assume every project is its own group, so you currently have x amount of groups, where x is the number of projects in the dataset. We also set a minimum similarity value of 0,75, we call this y.
2. Grab the highest similarity value.
3. If this value is bigger than y, then group the 2 corresponding projects together. Meaning x will decrease by 1. If this value is smaller than y, return the groups.

You can look at Figure 1 for an overview of how the algorithm operates.

3.2 Overview of the algorithm

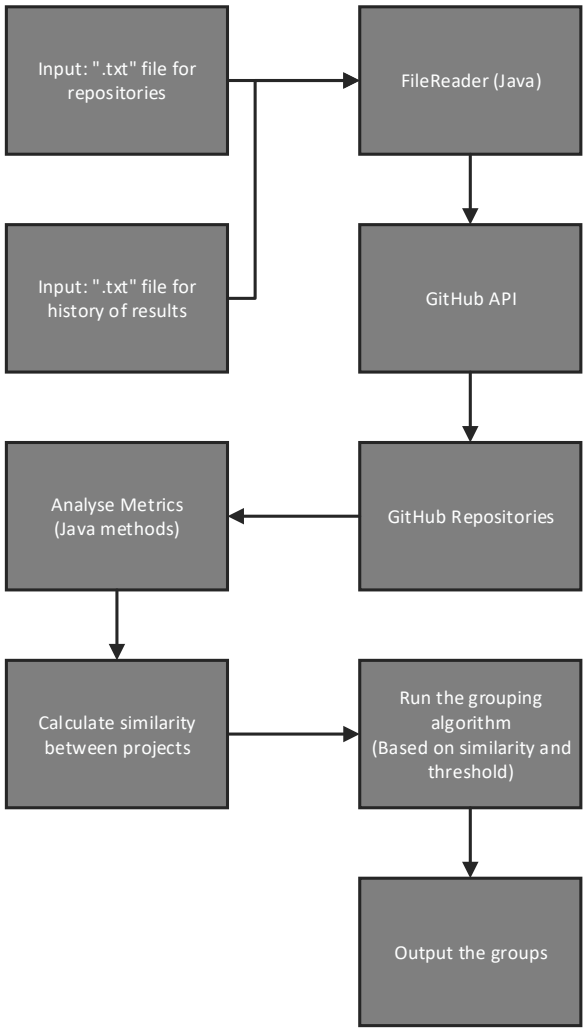

Figure 1: Algorithm overview

3.3 Choice of metrics

GitHub projects consist of a lot of metrics to define the GitHub repositories. But not all of them have to do with Interactions and Activities. In this section we will dive into which metrics are encompassed by interactions and activities.

We start by looking at some other research papers and what they have done. For example the CrossSim tool. which is a tool based on the OSS ecosystem: *"The following relationships are used to build graphs representing the OSS ecosystems and eventually to calculate similarity by means of the*

algorithm presented in the next section." [6]. After which they list the following metrics:

- the reliance of a project on a dependency
- is build by the same developer
- the stars in GitHub: *"we consider the star event in a broader scope in the sense that not only direct but also indirect connections between two developers is taken into account."*
- Similarities between co-development of other projects
- Source code identifiers
- If projects share code files

Now not all of these metrics are relevant to our cause. Namely, "the reliance of a project on a dependency", "Source code identifiers" and "If projects share code files", none of these metrics have anything to do with interactions and activities. Some metrics that are interesting are "is build by the same developer" and "the stars in GitHub".

Another example of a clustering or project grouping tool is CLAN [9]. They base their algorithm around semantic anchors like API calls. In other words, the metrics they use are based on code and not on interactions and activities and so are not useful to us.

We can also look at RepoPal. They use three heuristics to analyse repositories: Simmilarity in readme files, repositories starred by users of similar interests are likely to be similar, repositories starred together within a short period of time by the same user are likely to be similar. The last two metrics are interesting to us, since they indicate something about the interactions of people with the repositories.

Considering all of the above, we make a list of all the metrics that we deem to be part of interactions and activities.

Number of pull requests, Number of issues, Number of commits, Number of collaborators, Number of Releases, Number of branches, Number of Stars, Number of Watchers, Number of Forks, Time between pull requests, Time between commits, Size of pull requests, Size of commits

Side note: We left "repositories starred by users of similar interests are likely to be similar", "repositories starred together within a short period of time by the same user are likely to be similar" and "are the repositories build by the same developer" out, because they can't be analysed to find correlation by the Kendall Tau Correlation algorithm. This is because the Kendall Tau algorithm only works on numbers or variables which have an order or rank to them.

3.4 Similarity

Before we actually start our experiment, we still need to define when a project is similar. We do this based on a simple calculation explained in steps below:

1. Pick a pair of repositories.
2. For that pair calculate the difference between every metric.
3. For each metric calculate the maximum difference in the entire dataset.
4. Divide the difference between every metric by the maximum difference for those metrics.
5. Add the previous values together and take the summary of them.
6. That is your similarity value.

So, with this method, you define similarity relative to the dataset. For example when we have two basic datasets containing 3 values: set 1:{1, 5, 20} and set 2:{1, 5, 80}. In this example, with our method, in set 1, 1 and 5 are relatively less similar to each other than in set 2. Although this seems counter intuitive, we actually did this very much on purpose. Because when you would for example have a large group of very similar projects, you would still want to be able to divide the projects into groups. So to summarize, defining similarity in the way described above, will work for an ever growing dataset.

Why did we pick this way of defining similarity

Normalization within the Dataset: By calculating the difference between each metric for a pair of repositories and then normalizing it by the maximum difference within the entire dataset, we essentially scale the differences relative to the variability present in the dataset itself. This ensures that the similarity measure isn't biased by the absolute values of the metrics but rather by how those values compare to others in the dataset.

Flexibility for Diverse Datasets: In datasets where projects vary significantly in their metrics (such as different sizes or scopes), using relative differences allows for a more flexible definition of similarity. This means that even if projects have vastly different absolute values for certain metrics, they can still be considered similar if their relative differences are comparable.

Scalability with Dataset Growth: As we mentioned, this method is suitable for ever-growing datasets. Since the similarity calculation is based on the maximum differences within the dataset, adding new projects with different metrics or values won't fundamentally alter the similarity calculations for existing projects. This scalability is essential for datasets that are continuously expanding, such as those in dynamic environments like GitHub.

Intentional Design for Grouping: By deliberately designing the method to prioritize relative differences over absolute values, we ensure that even in cases where projects are very similar, there's still enough variability to differentiate and group them effectively. This is crucial for clustering algorithms or categorization tasks where finding distinctions between similar items is necessary.

3.5 Research question 1

GitHub projects encompass an extensive array of metrics, capturing various facets of development, collaboration, and community engagement. The abundance of metrics, necessitates a systematic approach to distill meaningful insights.

Since there are endless metrics to choose from while analyzing GitHub projects, we need to make a selection which we deem the most useful when comparing different projects.

For our first experiment, we want to answer the following question: "which attributes that are related to interactions and

activities would be suitable when comparing two open-source projects." We do the following steps to identify and compare metrics related to interactions and activities in open-source projects.

Firstly, we define a preliminary list of metrics, encompassing various aspects such as pull requests, issues, commits, collaborators, releases, branches, stars, watchers, forks, and more. See 3.3.

For data collection, 80 open-source projects are arbitrarily selected for comparison (You can find our dataset at [5]. To pick these projects, we use Github search tool: https://seart-ghs.si.usi.ch/ . This is a search tool called SEART made as part of a research paper [4]. Data for each metric is extracted from the GitHub repositories of the chosen projects over a relevant time period using the GitHub API and the wrapper [1].

We then write an algorithm that calculates the Kendall Tau correlation [8] between each of the metrics.

Next, we analyse the results of our algorithm and try to find the most expressive metrics to identify github repositories. We do this by trying to find correlations between the different metrics and ultimately elimination a couple metrics.

3.6 Research question 2

For our second experiment we want to answer the question: "Does grouping GitHub repositories based on inter-action and activities bring the same or different results as compared to grouping on different metrics?" We do this in the following way:

First off, we pick an algorithm made by someone else which groups GitHub repositories. In this experiment we will look at the CrossSim tool [3]. We chose the CrossSim tool, since we were looking at tools that use different metrics to find similar repositories than our algorithm. What also makes the CrossSim tool interesting to test against, is that they released their results as well as their dataset that they tested against.

Then we will look at the results of the CrossSim tool made by running their algorithm with their dataset. The results are in the form of groups of repositories. From these results we arbitrarily pick a couple of groups and use these groups of repository names as our dataset.

Furthermore, because our algorithm takes a while to analyse repositories, we chose to not use the full results of the CrossSim tool, but just a small subsection. You can see the repositories that we chose in table 1. You can also find our datasets on [5].

Group 1	Group 2	Group 3
1. links	7. asakusafw	13. webdrivermanager
2. jena-sparql-api	8. Ivory	14. learn_crawler
3. appstart	9. makela	15. selenium-standalone-server-plugin
4. jsonhome	10. lclueweb	16. selenograph
5. AutoSPARQL	11. Hive-mongo	17. selenium-cucumber-java
6. SPARQL2NL	12. msgpack-hadoop	18. webdriverextensions

Table 1: CrossSim results.

Then we will run our algorithm on that dataset, from here, we want to have 2 sets of results. The first set being the similarity values between the different result groups after we have ran our algorithm on them. So is group 1 similar to group 2 in terms of Interactions and Activities? And secondly, we want to know the similarity values from the projects inside of the groups. So are the projects that are grouped by the CrossSim tool, similar in terms of Interactions and Activities?

Next up, we will compare the two results against each other. There are 3 possible outcomes:

1. **The similarity inside of the individual groups is higher than the similarity between the groups.** In this case you can state that the CrossSim algorithm and our own algorithm lead to similar results.
2. **The similarity inside of the individual groups is lower than the similarity between the groups.** In this case you can state that the CrossSim algorithm and our own algorithm lead to different or even opposite results.
3. **The difference between similarity between the groups and the similarity inside of the individual groups isn't clear.** In this case you can state that the CrossSim algorithm and our own algorithm lead to different results.

4 Results

4.1 Research question 1

For our first experiment, we aim to identify key interaction and activity metrics for comparing open-source projects. We compile a metric list including pull requests, commits, and more. After selecting 80 projects using a GitHub search tool, we extract data via the GitHub API. Utilizing Kendall Tau correlation, we get the following results.

The results of correlation analysis between various attributes of open-source projects are presented in Figure 2. The first row and the first column of this table represent the pair of metrics. The numbers in the table represent the correlation of that pair of metrics. Here we can see which metrics have a strong correlation to each other and which metrics are more expressive. When the number you see in the table/figure is closer to 1 you could say that the pair of metrics that is expressed by that number is more positively correlated than a pair of metrics with a correlation closer to 0. The same can be said for numbers closer to -1, these metrics are more negatively correlated than pairs of metrics with a correlation closer to 0.

The highest correlations are between:

- number of issues and number of pull requests (0,77655)
- number of collaborators and number of pull requests (0,71070)
- number of collaborators and number of issues (0,69158)
- number of forks and number of number of stars (0,66745)
- number of commits and number of issues (0,66678)
- number of forks and number of watchers (0,65558)
- number of commits and number of pull requests (0,60765)

Metrics	num_pull_requests	num_issues	num_commits	num_collaborators	num_releases	num_branches	num_stars	num_watchers	num_forks	sched_pull_requests	sched_commits	size_pull_requests	size_commits
num_pull_requests	1,00000	0,77655	0,60765	0,71070	0,50131	0,51513	0,40586	0,42369	0,36481	-0,16539	-0,38893	-0,27341	-0,35949
num_issues	0,77655	1,00000	0,66678	0,69158	0,56352	0,49424	0,41519	0,43064	0,35259	-0,11682	-0,38897	-0,22144	-0,34030
num_commits	0,60765	0,66678	1,00000	0,59081	0,46509	0,53809	0,36897	0,35052	0,27412	-0,05706	-0,57547	-0,06748	-0,30900
num_collaborators	0,71070	0,69158	0,59081	1,00000	0,47460	0,45563	0,37340	0,46731	0,36096	-0,03730	-0,32445	-0,22351	-0,39752
num_releases	0,50131	0,56352	0,46509	0,47460	1,00000	0,34158	0,33577	0,32186	0,28426	-0,08584	-0,30739	-0,16285	-0,25385
num_branches	0,51513	0,49424	0,53809	0,45563	0,34158	1,00000	0,28287	0,28538	0,16403	-0,03339	-0,28254	-0,10297	-0,23003
num_stars	0,40586	0,41519	0,36897	0,37340	0,33577	0,28287	1,00000	0,59356	0,66745	-0,03783	-0,21737	-0,09860	-0,20344
num_watchers	0,42369	0,43064	0,35052	0,46731	0,32186	0,28538	0,59356	1,00000	0,65558	-0,08771	-0,21809	-0,15473	-0,23226
num_forks	0,36481	0,35259	0,27412	0,36096	0,28426	0,16403	0,66745	0,65558	1,00000	-0,02603	-0,15516	-0,16725	-0,21128
sched_pull_requests	-0,16539	-0,11682	-0,05706	-0,03730	-0,08584	-0,03339	-0,03783	-0,08771	-0,02603	1,00000	0,11573	0,24456	-0,12657
sched_commits	-0,38893	-0,38897	-0,57547	-0,32445	-0,30739	-0,28254	-0,21737	-0,21809	-0,15516	0,11573	1,00000	0,04140	0,15889
size_pull_requests	-0,27341	-0,22144	-0,06748	-0,22351	-0,16285	-0,10297	-0,09860	-0,15473	-0,16725	0,24456	0,04140	1,00000	0,38938
size_commits	-0,35949	-0,34030	-0,30900	-0,39752	-0,25385	-0,23003	-0,20344	-0,23226	-0,21128	-0,12657	0,15889	0,38938	1,00000

Figure 2: Results experiment 1, correlation between metrics

4.2 Research question 2

For the second experiment, we needed to pick some groups of repositories from the results given by the CrossSim tool [3]. We chose 3 distinct groups with repositories with entirely different subjects. The repositories we chose were seen in 1

Running our algorithm on these repositories led to the following similarity matrices in figures 3, 4, 5 and 6:

	Group 1	Group 2	Group 3
Group 1	1,00	0,97	0,91
Group 2	0,97	1,00	0,93
Group 3	0,91	0,93	1,00

Figure 3: Similarity between groups

	1	2	3	4	5	6
1	1,00	0,94	0,90	0,95	0,90	0,90
2	0,94	1,00	0,91	0,90	0,86	0,92
3	0,90	0,91	1,00	0,93	0,85	0,85
4	0,95	0,90	0,93	1,00	0,91	0,84
5	0,90	0,86	0,85	0,91	1,00	0,74
6	0,90	0,92	0,85	0,84	0,74	1,00

Figure 4: Similarity inside group 1

	7	8	9	10	11	12
7	1,00	0,90	0,97	0,61	0,95	0,97
8	0,90	1,00	0,97	0,53	0,89	0,90
9	0,97	0,97	1,00	0,53	0,96	0,98
10	0,61	0,53	0,53	1,00	0,63	0,58
11	0,95	0,89	0,96	0,63	1,00	0,95
12	0,97	0,90	0,98	0,58	0,95	1,00

Figure 5: Similarity inside group 2

	13	14	15	16	17	18
13	1,00	0,84	0,98	0,33	0,99	0,94
14	0,84	1,00	0,85	0,18	0,98	0,79
15	0,98	0,85	1,00	0,32	0,99	0,93
16	0,33	0,18	0,32	1,00	0,33	0,39
17	0,99	0,98	0,99	0,33	1,00	0,92
18	0,94	0,79	0,93	0,39	0,92	1,00

Figure 6: Similarity inside group 3

Where the numbers on the first row and the first column depict the projects and the numbers in the other rows and columns depict the similarity value.

In Figure 3, you can see that the similarity between the groups is high with a similarity value of 0,91 or more. Furthermore in Figure 4, you can see that the similarity inside of group 1 is also high with 1 pair of projects having a lower similarity but still not under 0,74. In Figure 5 you can see that all pairs in group two have a high similarity, except for pairs with project number 10 (lclueweb). The same phenomena can be seen in the similarity matrix of group 3. In Figure 6 all pairs of projects have a high similarity, except for pairs with project number 16 (selenograph).

5 Discussion

Main Research Question: Can we group GitHub projects based on interactions and activities?

Our findings reveal distinct clusters among GitHub projects sharing attributes related to interactions and activities. This signifies the influence of collaboration on project categorization.

Research Question 1: Which attributes that are related to interactions and activities would be suitable when comparing two open-source projects?

We explore a range of attributes associated with interactions and activities within GitHub repositories. These may include but are not limited to commit frequency, pull request volume, number of issues and collaboration intensity. By analyzing these attributes, we aim to identify key indicators that effectively differentiate between open-source projects and contribute to their categorization within distinct clusters.

Research Question 2: Does grouping based on interaction and activities differ from other metrics?

While interactions and activities contribute significantly to grouping, alternative metrics like code quality or project size might yield divergent results. This emphasizes the multidimensional nature of GitHub repositories and underscores the importance of considering various metrics in the analysis of GitHub projects.

In the following sections, we delve into specific findings and implications for each research question, providing a nuanced understanding of GitHub project grouping based on interactions and activities.

5.1 Research question 1

For our first experiment, we wanted to make a list of the most valueable or the most expressive metrics from our proposed

list of metrics. These metrics were: *Number of pull requests, Number of issues, Number of commits, Number of collaborators, Number of Releases, Number of branches, Number of Stars, Number of Watchers, Number of Forks, Time between pull requests, Time between commits, Size of pull requests, Size of commits*

To analyse these metrics, we calculated the Kendall Tau Correlation values for a set of 80 projects. From these calculations we got the results seen in the Results section with the highest correlations being between:

- number of issues and number of pull requests (0,77655)
- number of collaborators and number of pull requests (0,71070)
- number of collaborators and number of issues (0,69158)
- number of forks and number of number of stars (0,66745)
- number of commits and number of issues (0,66678)
- number of forks and number of watchers (0,65558)
- number of commits and number of pull requests (0,60765)

Since these metrics are the most correlated metrics, we can start to remove some of these metrics from our result set. Starting with the number of issues, which is highly correlated to pull requests. We will also remove number of collaborators, since this is the second highest correlation in the analysis. Since now both number of collaborators and number of issues are removed from the result set, we don't need their correlation value. Next up in the list is number of stars, number of stars is highly correlated to number of forks, and because of this we will be removing this metric as well. Next to number of stars, number of watchers also has a high correlation to number of forks and will be removed. And last, since number of commits is highly correlated to number of pull requests as well, we will be removing this metric as well.

This leaves us with the following resulting list of metrics: *Number of pull requests, Number of Releases, Number of branches, Number of Forks, Time between pull requests, Time between commits, Size of pull requests, Size of commits*

Since now the most correlated metrics are removed, we can say that this is our most expressive list of metrics and because of that reason, also the most suitable when when comparing open-source projects, Answering our first Research Question being *Which attributes that are related to interactions and activities would be suitable when comparing two open-source projects?*

This experiment, however, could still be improved. Earlier in this document we discussed a which metrics we would test against and which metrics we would not test against. Also we mentioned that there are endless metrics to chose from when analysing GitHub projects. So the choice of the test set of metrics, although explained carefully in the methodology, is a liability and there might still be other metrics that could be part of the "perfect" set of metrics. Another possible parameter that could have negatively influenced our results, was the size of our test set of GitHub projects. While we chose a set of 80 project, a larger set of projects could further solidify the results of the correlation tests.

5.2 Research question 2

Our seconds experiment was a bit different. We looked at the results of the CrossSim tool [3]. From these results we picked 3 groups with repositories made by running the CrossSim tool on a set of GitHub projects. We now had 3 groups of, according to the CrossSim algorithm, similar projects. After this, we wanted to test whether our algorithm would "agree" with the CrossSim algorithm by running our algorithm on the results of the CrossSim algorithm. From there we want to look at whether the similarity inside of the groups is higher lower or not clear as compared to the similarity between the groups according to our algorithm.

The results we got out of running our algorithm on the repositories above were in the form of 4 similarity matrices. The first matrix 3 depicts the similarity between the different groups. And the next 3 matrices (4, 5, 6) depict the similarity between the projects inside of the groups.

When analysing our results we needed to account for 3 different scenario's:

1. The similarity inside of the groups is higher than the similarity outside of the groups.
2. The similarity inside of the groups is lower than the similarity outside of the groups.
3. The difference between similarity between the groups and the similarity inside of the groups isn't clear.

Looking at the similarity matrices, we can conclude that we are looking at scenario 3. The difference in similarity between the inside of the groups versus the similarity between the groups themselves, is not evident. Since the similarity between the groups is still very high on average 0,94 and the similarity inside the groups is barely any lower, 0,81. However the last average may be a bit fallacious, since both in group 2 and 3 there is one project that doesn't have a high similarity with any of the other projects in those respectful groups.

This means that we can state that our algorithm leads to different results as compared to the CrossSim algorithm. This could be because there is little to no correlation between the metrics used by the CrossSim algorithm and the metrics used by our own algorithm.

After having analysed our results, we can answer the research question *Does grouping GitHub repositories based on inter-actions and activities bring the same or different results as compared to grouping on different metrics?"* The answer being that grouping GitHub repositories based on Interactions and activities yields different results as compared to grouping on different metrics.

Difference in results between our algorithm and an other algorithm might actually be a good thing. Since that means that we could combine the two algorithms to improve the grouping algorithm in a way that it then not only recognizes if projects are of the same subject, but also if projects are developed in a similar way.

This experiment, however, didn't go perfectly. We ran into one big issue and that was time. Our algorithm was fairly slow, and we didn't have the computational capacity to let the algorithm run for days on end. This is why we opted to use a small part of the dataset that the CrossSim tool used.

5.3 Future work

Looking forward, our research sets the stage for future projects by demonstrating how GitHub projects can be grouped based on their interactions and activities. Moving forward, a natural progression would involve creating a user-friendly software tool that goes beyond our current focus on these specific aspects.

This tool could be really helpful, especially for businesses looking to simplify how they discover projects. By considering a wider range of factors, the tool has the potential to be a valuable resource for finding projects that are similar. For businesses, this means they can more easily find projects that are relatable, giving them a chance to adopt successful practices from similar projects. This contribution is especially important as it has the potential to improve efficiency and encourage the sharing of knowledge in different professional fields.

Furthermore, some more research can be done on this field as well. In other papers about grouping GitHub project, or projects in general, there has mostly been talk about finding similarity in the sense of the subjects of these projects. But as we have shown in this paper, interaction and activities could also be interesting and could prove to be very useful. So it might be very useful to have some research in combining these grouping algorithms and making a search-engine for projects, that cannot only identify similar projects in terms of subject, but also in terms of how they were build.

Another piece of research that relates to the subject of this paper, that could be interesting, would be researching if it is possible to find relations between metrics from a interactions and activities point of view, and metrics from a source code point of view, or a dependency point of view.

6 Conclusions

In conclusion, our research set out to investigate the feasibility of grouping GitHub projects based on interactions and activities, with a focus on identifying suitable attributes for comparison and exploring the differences between this approach and other metrics-based grouping methods.

Our findings have demonstrated that distinct clusters can indeed be identified among GitHub projects by analyzing attributes related to interactions and activities. This underscores the significance of collaboration in categorizing projects effectively. Through our exploration of various attributes such as commit frequency, pull request volume, and collaboration intensity, we have identified key indicators that contribute to project categorization within distinct clusters.

Regarding our first research question, which focused on identifying suitable attributes for comparison, we conducted experiments to determine the most valuable metrics from a proposed list. By analyzing correlation values, we narrowed down our list to the most expressive metrics, providing insight into which attributes are most relevant when comparing open-source projects.

For our second research question, which examined the differences between grouping based on interactions and activities and other metrics, we compared our algorithm's results with those of the CrossSim tool. Our findings indicated differences in results, suggesting that grouping GitHub repositories based on interactions and activities yields distinct outcomes compared to grouping on different metrics. However, this variance presents an opportunity for future research to combine algorithms and improve grouping methods, potentially leading to more comprehensive project categorization.

Looking ahead, future work could involve the development of user-friendly software tools to facilitate project discovery based on interactions and activities. Additionally, further research could explore the combination of grouping algorithms to create a search engine for projects that considers both subject similarity and development methodology. Moreover, investigating the potential correlations between metrics related to interactions and activities and those related to source code or dependencies could provide valuable insights into project development dynamics.

In essence, our research contributes to a deeper understanding of how GitHub projects can be grouped, paving the way for future advancements in project categorization and facilitating more efficient knowledge sharing and collaboration within professional fields.

In conclusion, our study addressed the main research question of whether GitHub projects can be grouped based on interactions and activities. The findings reveal distinct clusters among projects that share attributes related to collaboration, emphasizing the influence of collaborative efforts on project categorization.

7 Responsible Research

In the course of developing this research project, we prioritize ethical considerations at every stage of research and development. Our dedication extends to safeguarding user privacy, ensuring robust data security measures, and promoting inclusivity. The software aligns with open source principles, and we proactively explore methods to reduce its environmental footprint. Continuous monitoring and improvement are integral to our commitment to evolving ethical standards, contributing to the creation of a responsible and sustainable digital environment.

For our appendix, code and other usefull bits of information, we use Zenodo. Zenodo is a versatile digital repository established as part of the European OpenAIRE initiative and managed by CERN. We use this software because it is funded by the European Commission and guaranteed to last for at least another 10 years. Upon submission, each item is assigned a unique and persistent digital object identifier (DOI), facilitating easy citation and retrieval of stored content.

Additionally, we prioritize transparency in our data collection and processing methods, providing users with clear and accessible information about how their data is used and protected.

References

[1] Github api documentation. https://github-api.kohsuke.org. Accessed on 08-01-2024.

[2] Jailton Coelho, Marco Tulio Valente, Luciana L. Silva, and Emad Shihab. Proceedings of the acm/ieee inter-

national symposium on empirical software engineering and measurement (esem '18). Conference Abbreviation '18, pages 1–10, New York, NY, USA, October 2018. ACM.

[3] CrossMiner. Crosssim: Cross-project code similarity detection tool. https://github.com/crossminer/CrossSim. Accessed on 08-01-2024.

[4] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. Sampling projects in github for msr studies. *arXiv:2103.04682v1 [cs.SE]*, March 2021. Available at https://arxiv.org/abs/2103.04682v1.

[5] Rowan de Bruin. Datasets for research question 1 and 2, 2024. Available at https://doi.org/10.5281/zenodo.10575460.

[6] R. Guerraoui, N. Kuznetsov, and M. Monod. Crosssim: A tool for identifying similarities between open source projects. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 511–515, Sept 2018.

[7] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D.M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.

[8] M. G. KENDALL. A NEW MEASURE OF RANK CORRELATION. *Biometrika*, 30(1-2):81–93, 06 1938.

[9] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, June 2012. IEEE.

[10] Y. Zhang et al. Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23, Klagenfurt, Austria, February 2017. IEEE.