Finding your digital sibling: which other GitHub projects are similar to yours?

Finding similar repositories based on the available documentation

Alexandru Catalin Turcu[1]

Supervisor(s): Dr. Ing. Sebastian Proksch[1]**, Shujun Huang**[1]

[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: Alexandru Catalin Turcu
Final project course: CSE3000 Research Project
Thesis committee: Dr. Ing. Sebastian Proksch, Shujun Huang, Dr. Julia Olkhovskaya

Abstract

This paper aims to study the importance of considering the documentation side of GitHub repositories when assessing the similarity between two or more applications. Readme and Wiki files, along with Comments from the source files are the dimensions proposed to be analyzed through our methodology and experiments. We propose a pipeline that first extracts text fragments from these dimensions and then applies Natural Language Processing techniques to further prepare our data for evaluation. To gather a similarity score, we first vectorize our processed data with TF-IDF and then use cosine distance to obtain the score. Combinations of the three dimensions, ranging from using only one dimension to using all of them, are considered throughout our study. Moreover, additional information has been extracted from the plain text, such as referenced URLs and License usage, the similarity of which was calculated using Jaccard distance. Two experiments were performed. The first one aims to observe the behavioral tendencies of our methodology applied to a small dataset, while the second one aims to validate our results. By evaluating them, we found sufficient data that supported our presented conclusion: documentation represents a valuable asset in gathering a pool of similar applications.

1 Introduction

During the year 2022, GitHub has reported the creation of more than 52 million new open-source projects [1]. This establishes GitHub as one of the most popular choices for hosting and supporting projects throughout their entire development cycle. These repositories can share valuable insight through the available code and documentation, crucial first-hand experience for an individual searching for a role model project. However, this process can become overwhelming due to the sheer number of available sources.

For this reason, an automated method for finding and extracting similar repositories has been the focus of numerous research experiments. In the past years, *Zhang et al.* [2] firstly considered the usage of Readme files as one of the dimensions for evaluating the similarity between repositories in one of the baseline tools in the domain. For this dimension, the text is preprocessed by using Natural Language Processing (NLP) [3] procedures: removing the stopwords and then applying tokenization, and then converting into a vector of weights based on each word's TF-IDF (a measure of the importance of each word in a document from a collection of documents [4]). As a different approach, *Nguyen et al.* [5] proposed an automated tool, CrossSim, for mining repositories and evaluating their similarity based on graph representation that relates developers to their code. *Auch et al.* [6] has recently completed a comprehensive overview and analysis of various methodologies used to calculate similarity, published before 2020. This analysis serves as a valuable resource to understand these methodologies' evolution and relevance in modern times. However, most of these tools used in the industry do not utilize valuable available data from the documentation side (besides the Readme files) of the GitHub repositories, such as the Wiki pages accompanying projects and comments written by the developer teams in the source code files.

Therefore, this paper aims to answer the following research question:

"How similar are GitHub projects that share attributes on the documentation side?"

In other words, the aim is to delve into the assessment of the similarity between repositories based on their documentation, with emphasis on their Readme files, associated Wiki, and code comments. This question can be further divided into the following sub-questions:

RQ 1 What segments of each documentation dimension are the most relevant for finding similarities?

RQ 2 Which branch (dimension) or combination outputs the best results?

RQ 3 Should the lack of documentation make two projects similar or not?

In order to successfully answer the main research question, we first need to define the meaning of similar repositories. Later, a methodology to extract the required data will be introduced: an algorithm that first extracts the raw text from the Readme file, Wiki, and comments in three different files, which is then processed using NLP techniques and, lastly, a similarity score, which needs to be interpreted, is outputted from the comparison of the data of each two different repositories. In addition to the previously described method, for each sub-question, a specific methodology will also be defined to facilitate their evaluation, which will contribute to the solution of the research problem. Another issue that will be discussed further in this paper centers around selecting a set of suitable test repositories that ideally would contain both active and inactive projects, as well as personal and professional ones. We believe that by splitting our research question into these three concentrated sub-questions will help us focus on the key aspects that can be used to justify our conclusions more effectively.

This question is part of a larger research topic that aims to analyze the impact of various dimensions when comparing repositories. The common goal is to combine the studied dimensions into a single tool. Therefore, the dimensions of each research topic must be similar, and no dimension should dominate the others.

To justify our study, two types of experiments were considered for our process. The first one concentrated on the optimization of our methodology as well as providing the direction of our approach, and then the latter was used to validate our findings. After that, we observed that the accuracy of our experiments increased based on the amount of available extracted data (the usage of two or more dimensions involving text from Readmes, Wikis, and Comments). Alongside this, we concluded that using a single aspect of the documentation did not yield satisfactory results,

whereas mixing the extracted text from multiple sources is a valuable asset in identifying similar repositories.

Continuing this paper, we propose the following structure, divided into multiple chapters: Section 2 defines the terminology and tools utilized. Section 3 will go into detail regarding the literature. Section 4 will define the used methodologies, and Section 5 explains the experiment setup. Section 6 evaluates the results of the proposed experiments, while the conclusion and future work can be found in Section 7. Finally, Section 8 shortly reiterates our findings, while Section 9 presents the responsible research principles that were followed.

2 Background

Before delving into the proposed methodology, we want to provide additional context for the tools and technologies used.

In general, to facilitate the procedure of text mining, ordinary terms that bring little to no helpful information are removed. As a starting point for our list of stopwords, we considered the dictionary available through the NLTK[1] package.

Tokenization represents a method of splitting a text into distinct words and words into letters. In our paper, we always considered the first use case of tokenization [7].

Lemmatization is a process of reducing words to their original form, similar to how stemming works, but in addition to this technique, it also considers the context in which a term is used [8].

We constructed a dictionary of license names and acronyms, sourced from the SPDX database[2], that was used to recognize them in the extracted but not yet processed text.

Regarding code comments, we referred to Javadoc[3] on several occasions during this paper. This represents the addition of tags to comment chunks that facilitate the automatic creation of documentation for a package.

3 Related Work

Currently, multiple tools such as **Repopal** [2], **CrossSim** [5], or **CLAN** [9] are used to detect similar GitHub repositories or software, each one proposing different methodology concepts. CLAN has been considered since its release as the state-of-the-art process of identifying similarities between Java applications. It computes the index score based on API calls, which are given weights inversely proportional to their popularity. Moreover, the approach considers frequent sets of APIs, which leads to a higher precision of clustering applications.

On the other hand, Repopal investigates three completely different heuristics: the content of Readme files, starred projects by similar users, and repositories starred together in a short period. This strategy was compared to the CLAN approach and was concluded to outperform the older strategy on crucial metrics such as precision. It is of utter importance for our research how *Zhang et al.* handled documentation files, namely applying NLP procedures in eliminating stop words and reducing remaining words to their root form.

CrossSim proposes a graph-based approach that has validated its results using Repopal and CLAN. This tool concentrates on the interaction and relationship between the developer and the source code but does not cover the documentation side. However, this tool represents a valuable asset to consider for validating our results.

An important role in our paper is played by the process of comparing text documents. Consequently, evaluating the chosen methodology highly depends on the algorithm chosen for comparing such documents. *Usino et al.* [10] proposed text clustering by using K-Means on the calculated cosine distance of vectorized text. Typical NLP techniques are applied to remove undesired data. While this methodology is used for detecting plagiarism, we believe a similar approach could be used for our aim.

To complete our research, we had to establish a suitable dataset of repositories for our studies, a factor that can represent the difference between an accurate or inconclusive set of results. The tool **reaper** [11] has been proposed to identify maintained and reliable software based on multiple practices that can be observed on their GitHub pages. This includes extensive documentation, test coverage, and management of issues.

4 Methodology

To answer our research question of finding similarities between repositories, we propose the following heuristics: one common methodology of extracting the data required to perform the analysis and three distinct methodologies, one for each of the derived research questions.

Similarity Definition

In order to manage the task of finding similar repositories and clustering them based on shared topics, we had to set up a definition for similar projects. Therefore, two or more repositories were considered similar if one of the following criteria is met:

1. Their goal is to complete the same task, with no emphasis on the technology.
2. Their end goal differs, but they have a common methodology to get there.

Experimental dataset

Even though GitHub offers a plethora of open-source repositories, we had to manually identify suitable projects for our study. We established a series of metrics that needed to be respected by each project to be considered in our experiments. We primarily selected repositories that were actively maintained at the moment of extraction and displayed at least a descriptive introduction in their Readme file. We decided to involve projects with different levels of process management, from projects that required pull requests to be accepted by active software maintainers to projects that did not even use an active issue table.

Our goal was to define a dataset covering a broad number of different topics, in which we expected the similarity

[1] https://www.nltk.org/

[2] https://spdx.org/licenses/

[3] https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

distance between repositories that would represent the same topic to be significantly lower than when compared to a similarity distance to a project in a distinct cluster. Moreover, language was used as an additional metric, which must be English for the documentation side of each project.

Only the wikis directly available for a repository on their GitHub page were considered for this research. While this has limited the available data, extracting data from external websites represented an impediment due to different formats of displaying the information. Another issue we considered was that an unknown amount of repositories did not even offer a wiki page. We aimed to construct a dataset that contained a sufficiently high percentage of repositories implementing a wiki. Additionally, we limited ourselves to proposing mostly projects written in Java to further restrict the acceptance criteria.

To validate our results, we considered the usage of an additional and broader dataset that has already been evaluated by a state-of-the-art tool, namely CrossSim [12]. The two datasets were not combined in any way to keep test and evaluation data distinct. The defined heuristics were not applied to this dataset.

Data extraction & processing

The documentation side of each repository is split into Markdown for Readmes and Wikis and source code files, in which comments are present. To extract the data, an iterative approach is proposed, in which the text from each dimension is extracted into locally saved files.

In Markdown files, there are two types of tags: HTML ones and their basic syntax, from which we consider either the text that is outside any type of tags, text from within tags used to render it in a different format, or from captions of images or other figures. We considered these tags unimportant information and removed them from the resulting file.

To extract comments, we considered identifying the lines containing them using regular expressions. For this, we had to consider that developers use both in-line and multi-line comments interchangeably. Additionally, constructs that follow the Javadoc format present a multitude of additional tags that, if they were to remain in the extracted data, would alter the results.

In the case of wikis and comments, we expected to extract the data from multiple input files that would generate an abundance of output files. The text is merged into a single plain text file for each dimension to fix this potential issue.

Data processing is completed by following multiple NLP techniques that are required to remove any irrelevant information from the extracted text. We filter it using our enhanced stopwords dictionary, to which words from an ever-growing list of common terms used by software developers in descriptions are appended. Variable and function identifiers referenced in comments are considered out of the scope of our study and therefore removed if their format follows the rules of *camelCase*. Finally, symbols are identified and removed during the tokenization procedure, followed closely by data lemmatization, to obtain the base form of the words in context.

Similarity evaluation

To obtain a floating point number representing the similarity score, we decided to transpose our text data in a frequency vector that translates each word's importance in the context of all considered files containing processed text. This represented for us an ideal scenario to apply the TF-IDF algorithm, whose scope is precisely our need. We decided to further adjust the weights of certain selected tokens that can describe the purpose of an application. These words were identified in the vector by converting the hashed values back to their original form, and later, the adjusted weights were added to the vector.

The outcome is a very high-dimensional vector. Throughout our study, we decided to experiment with using a lower dimensionality vector that can be obtained by analyzing and keeping only the most significant values. We identified that Singular Value Decomposition (SVD [13]) could help us in reducing the effect that noise would have on our results, as well as representing a tested method to gather and use only the most relevant values.

With the data vectorized, we then calculated the similarity of every two repositories by using cosine similarity, represented by the division between the dot product of two frequency vectors and the multiplication between the magnitude of the vectors. The output is represented by a decimal number score between 0 - highly dissimilar and 1 - highly similar.

An overview of the extraction methodology can be visualized in Figure 1, while a lower-level description of our methodology is presented in Section 5, Experimental Setup.

RQ 1 methodology

The first research question aimed to determine the relevance of each dimension fragment, and to achieve this, we first analyzed a series of repositories to notice tendencies in documentation writing.

Throughout the dataset elicitation process, we have observed that besides the text used as a description in documentation, there were present a couple of URLs mainly used to acknowledge and cite sources, along with licenses that are utilized to regulate the usage of a particular open-source project in someone else's work. To identify them in an automated procedure, we had to rely on regular expressions to gather website links from the text and on the usage of a license dictionary between their names and acronyms to identify them accurately. The licenses and URLs were always considered from an unprocessed variation of the extracted data.

To facilitate the experimentation of the three distinct approaches, we decided to divide our data into three distinct fragments. From the original defined files that contain the text extracted from the Wikis, Readmes, and Comments, we filtered out and saved the new features in two additional files, one for each URLs and Licenses. These files combined the extractions from all original dimensions.

We applied a filtering process to the newly extracted features to ensure the uniqueness of each entry. To gather a similarity score between two repositories based on either URLs or Licenses we used Jaccard similarity, which

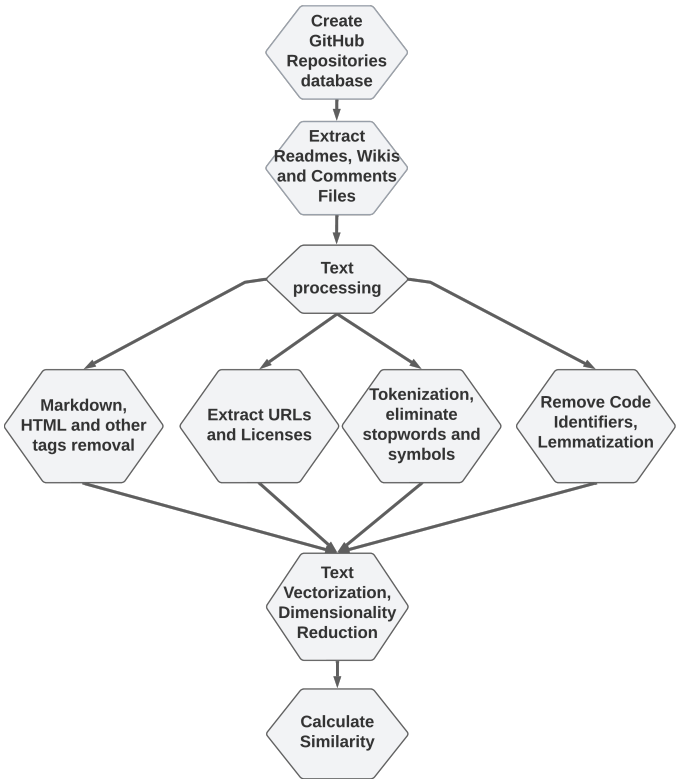

Figure 1: Flowchart of the proposed data extraction methodology

considers the cardinal number of the intersection of two sets divided by the cardinal of their union.

RQ 2 methodology
By considering three distinct components of repository documentation, we were free to explore the usage of seven different scenarios that involved using either one, two, or all three available dimensions. Our goal was to identify the best-performing scenarios in terms of the accuracy in marking pairs of repositories as similar. Therefore, we considered all the data sets that can be generated and evaluated the repositories' similarity based on every option. If we were to denote the three dimensions as R (Readme), W (Wiki), and C (Comments), the final list of possible combinations would be: *R*, *W*, *C*, *RW*, *RC*, *CW*, *RCW*.

RQ 3 methodology
By defining the previous methodology in which single components of documentation were compared, it is essential to detail the handling of cases when certain components were missing and to establish a correlation between these repositories. We want to emphasize that repositories that did not supply any documentation were not considered to take part in our dataset and, therefore, experiments. However, missing a single dimension of the three represents a common occurrence. We assigned an initialization value for each similarity score to handle this case. Throughout our experiments, we tried three different scores: 0 (highly dissimilar), 0.5 (neither similar nor dissimilar), and 1 (highly similar). We expected these scores to mainly influence the evaluation process during the comparison of Wiki files, which, based on our observations, represent the rarest dimension of documentation.

5 Experimental Setup

The study described in this paper is supported by source code written in Java 14 and Python 3.10. To successfully run an experiment, firstly the data had to be extracted from the selected repositories. The Readme is obtained through direct usage of API calls to GitHub. The contents are usually saved in a Markdown file containing both HTML and proprietary tags to facilitate the usage of images, for example. For our experiments, we extracted all the available plain text and the paragraphs encapsulated in the caption of the aforementioned tags.

Comments were extracted recursively from the folders of each repository by searching for files that had programming language extensions, with the better part of the experimental dataset being assembled out of Java-focused projects. To extract the text, we decided to use regular expressions (Regex) to identify lines or multi-lines of comments based on the designated symbols used to denote them. Initially, we encountered multiple issues related to both inline and blocks of comments. These issues resulted in chunks of code commands being added to our extracted file. Part of these issues were mitigated by refining our regular expressions to not only identify lines that contain comments but also accurately identify the type of comment that follows (i.e., independent comments or comments on the same line as code content) and its start and end points. For this, we had to increase the complexity of conditions by verifying whether the extraction of a comment was finalized before starting to extract a different one. Since we concentrated our attention on Java repositories, we defined a methodology that can directly identify and eliminate tags used in Javadoc to generate documentation pages by keeping a list of the available tags.

Lastly, Wiki pages cannot be accessed directly through the available APIs and must be downloaded on the local machine running the experiments. Since Wikis are mostly considered for online usage and editing, many file names contain symbols deemed illegal by the naming conventions on a machine running Windows, a problem that can be mitigated by running the experiment on a Unix-based operating system. Moreover, on GitHub, they are considered part of different repositories compared to the projects they are attached to. Consequently, to access them, we were required to append '.wiki.git' to the original link of the repository. A Wiki repository is often split into multiple Markdown files, from which the same method of extracting text, as in the case of Readmes, was used. Finally, the extracted text is merged into a single file.

We would like to acknowledge our colleague Juul Crienen for the development of a GitHub API wrapper[4] that simplified the process of handling files in the case of extracting comments.

[4]https://github.com/jcrienen/github-api-wrapper

All the extracted text went through our processing pipeline. URLs and Licenses have been extracted into different files and removed from the original ones. URLs were identified through the usage of regular expressions, while licenses were identified by their presence in a database offered by SPDX.

The following steps involved the removal of stopwords (both technical and English common words) from the extracted text, which is then tokenized to facilitate the removal of the punctuation symbols and the removal of code identifiers (for languages using camelCase practices). Finally, each word was stemmed into a nonarticulated form according to its context.

During the experiments, seven scenarios of combining our documentation dimensions are considered (combinations of size 1, 2, or 3 that can be obtained from Readme, Comments, and Wikis), and depending on the selected scenario, the processed contents are merged. We obtained a matrix containing similarity scores between each of the two repositories, calculated using cosine similarity on the vectorized data obtained through applying TF-IDF.

We proposed a more straightforward method to calculate similarities between the URL and License dimensions, in which unique entries are considered, and the scores are saved in a similar matrix, but using Jaccard similarity this time.

Two types of experiments were run to evaluate the proposed methodologies: The main objective of the first experiment is to answer the three proposed research sub-questions and collect valuable insights that can be visually analyzed for correctness. For this experiment, a manually selected and labeled dataset has been used to evaluate the performance of our approach. Due to the proposed heuristic of using K-means clustering for grouping similar repositories, we had to resort to using the SciKit-Learn [14] package provided function *adjusted_rand_score*[5], which calculates the accuracy of the labeling made by the clustering algorithm, adjusted for randomness. For this reason, the accuracy of each cluster might be misleading (and slightly different for each run), and a manual evaluation of the results was done.

The second experiment was focused on validating our methodology against the results outputted by the CrossSim tool, using a similar dataset adjusted for possible unavailable repositories. The study attached to the tool provides user evaluation for software flagged as similar, which will be considered for our validation. However, this scenario does not consider the importance of features such as URLs or Licenses. Moreover, no dimensionality reduction was applied to the data in this experiment. For the remaining scenarios considered in this experiment, we selected each of the first ten pairs of repositories that were flagged as similar, in decreasing order regarding their score.

A modular approach was preferred to ensure an efficient environment that can be modified straightforwardly, in which each of the three steps saved their results to external files. The usage of different programming languages represents a trade-off between the desired technology usage and the nature of the data to be processed: repository extraction can be handled by Java, while libraries such as SciKit Learn in Python are the de facto standards when processing large text documents.

A detailed procedure for running the experiments locally, in addition to the source code, can be found on the publicly available personal GitHub repository [15].

[5]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

6 Results

The results of the two proposed experiments provided meaningful insights that contributed to formulating our reasoning for the research questions. Even though the results of the analysis experiment produced a slightly different output at each run, we observed a tendency in the resulting graphical representation. Figure 2 illustrates an example of a calculated accuracy graph between the expected and the empirical findings. This graph contains all the considered initialization scores of similarity for a pair of repositories.

The validation experiment yielded results deemed most precious for establishing a theory on the importance of each scenario considered. Regarding the setup, we established the initial similarity score between each pair of repositories as 0, which is highly dissimilar.

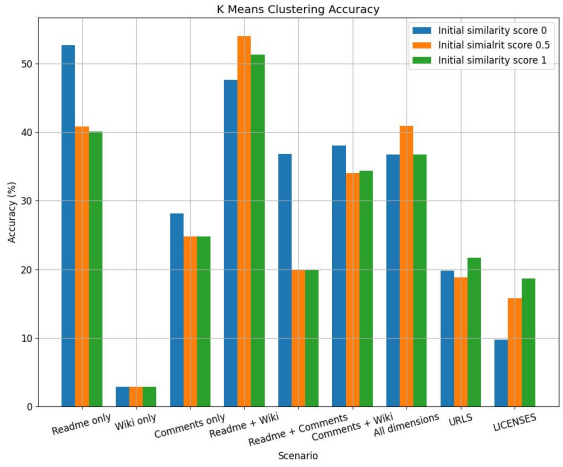

Figure 2: Accuracy of clustering for each combination of dimension: Readme, Wiki, Comments, along with URLs and Licenses, with different initial similarity scores

RQ 1 results

The first proposed research question explores the relevancy of different segments of each documentation dimension, which were split into three distinct categories: URLs, Licenses, and the remaining text. For each of the three categories, the data has been extracted into separate files, filtered to contain unique entries (in the case of URLs and Licenses), and processed stemmed tokens for the remainder.

By referring to Figure 2, it can be observed that the URL and License dimensions were outperformed drastically by the remaining combination (excluding Wiki, an aspect that was treated in a subsequent research question). These features did not cluster the repositories in a manner that is at least

similar to the established ground truth in comparison with the remaining scenarios.

We are confident that this bottleneck happened due to our similarity calculation methodology. Nevertheless, URLs and Licenses were not utilized in performing the validation experiment since we concluded that we could obtain more accurate results for our study by evaluating the remaining scenarios.

Therefore, we believe that according to our methodology, lemmatized sets of words represented the correct strategy for comparing repositories.

RQ 2 results

The second research question represents an attempt to identify the most accurate and complete set of documentation dimensions that would best associate similar repositories. For this purpose, we defined seven distinct scenarios that involved the three proposed dimensions: Readmes, Wikis, and Comments. The cases consisted of combinations of these dimensions of sizes ranging from one to three.

The evaluation of this research question highly benefited from the validation experiment. However, we would like to begin by presenting the observations we gathered from the first experiment's results.

In Figure 2, we can observe that the lowest accuracy-wise scenario happened during the clustering of repositories based only on the content of their Wiki pages. Moreover, out of the scenarios considering a single dimension, only the Readme one performed comparably to the scenarios implementing two or more sources for data.

The scenarios containing data from source code comments tended to produce similar levels of accuracy scores. To visualize the impact of this dimension over the remaining parts of the documentation, we analyzed the size, in terms of lemmatized words, of each. Figure 3 plots the distribution of each category, and it can be observed that the size of Comments dominated the others. Consequently, the extracted comments would behave similarly to their singular scenario when combining two dimensions due to the sheer amount.

Part of our dataset comprised repositories that are part of the Spring Framework projects[6] aimed at implementing multiple database technologies in their tool. These industry-based repositories are often sustained by a similar group of authors. By manually analyzing the resulting clusters from our experiments, we observed that this particular group of repositories is highly correlated according to our methodology, deemed neutral to other repositories that implement a database, and most often dissimilar to non-related repositories (such as open-sourced games). On the remaining part of the dataset, the results were not that straightforward, but a tendency to group similar repositories still existed (for example, even though the repositories that contained bots for Discord weren't all clustered together, there were pairs of at least two of them that ended in the same cluster).

The validation experiment confirmed the initial results regarding the only Wiki scenario: an unexpectedly high number of repositories had 1 as a similarity score. When analyzing the Wiki pages of the highlighted projects, we found the issue: all these Wiki pages had the placeholder text *"Welcome to the project wiki"*.

For the other six scenarios, 39 unique pairs of similar repositories were identified (out of the 60). Three of them have been marked during the user evaluation of the CrossSim tool (which only considered around 250 pairs) as highly similar, and one as mostly dissimilar. Due to the low number of user-evaluated pairs, we could not efficiently compare the outputs of the two algorithms, so we manually analyzed all resulting pairs by splitting repositories based on their topics, for which we applied our definition of similarity to the analysis.

The 'Readme only' scenario represented the worst-performing remaining dimension, where only two of the nine unique pairs considered resembled a similar topic. As a comparison, only three unique pairs were deemed highly dissimilar for the remaining scenarios. The most common topics shared by repositories marked by our heuristics as similar were related to data management, big data analysis, and mobile music players. Comparable with our previous experiment, we observed that projects that were sharing developer teams achieved a higher similarity score.

In the validation set, we observed that projects that have been forged or represent updated versions of a tool tend to achieve a high similarity score, according to our metrics. An overview of the validation experiment's manual evaluation is available in our online appendix hosted on the Zenodo OpenAIRE repository [16].

By analyzing the two experiment types, we can conclude that the best performances were obtained by scenarios that considered at least two different components. Specifically, with our methodology, we would recommend the usage of the Readme-Wiki scenario. The scenario that additionally included comments represented a slightly less accurate technique, and we would not recommend its usage yet. By analyzing the files with extracted text from Java source files, we identified specific key terms used in programming and concluded that we did not handle correctly commented code chunks, and as a consequence, they influenced the similarity calculations.

RQ 3 results

The last research question aimed to identify a way to handle the cases of missing documentation. We previously proposed a methodology where we experiment with the initialization of the similarity score between pairs of repositories: mark them as either dissimilar (a score of 0), neither similar nor dissimilar (0.5), or similar (1). For example, in the 'Wiki only' scenario, this value would be updated only when both repositories in a pair contained text extracted from a Wiki page. We first evaluated all the options in the analysis experiments, where we ran our methodology using the proposed initial similarity scores.

In these experiments, the clustering accuracy did not vary much throughout the cases, so the results were mostly inconclusive. However, during the validation experiment, where we ran our experiment only with an initial similarity score of 0, we observed an unexpectedly high number of

[6]https://github.com/spring-projects

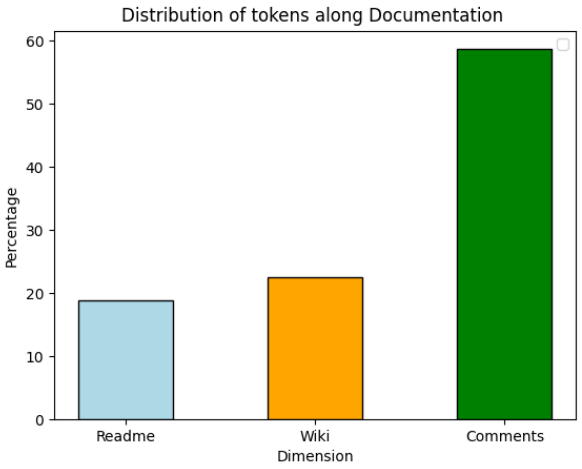

Figure 3: Distribution by the number of tokenized words in each considered main dimension of the documentation side. Comments represent $\sim 58\%$, Readmes $\sim 19\%$ and Wikis $\sim 22\%$.

pairs marked with a perfect similarity score in the Wiki-only scenario. By manually analyzing the content of the Wiki pages, we identified the issue: a large number still had the original placeholder template. Moreover, only 74 out of the 560 repositories in the validation dataset implemented a Wiki page, from which we gathered high numbers of pairs of unrelated projects with high similar scores and similar pairs with a score of 0. However, by deeming repositories with missing documentation similar, we would encounter more false positive cases. So, as a middle ground, we believe using a neutral initial similarity score of 0.5 is advised compared to assigning scores that would entail an extreme relationship status.

7 Discussion

By answering **RQ 1**, **RQ 2** and **RQ 3**, we were able to construct the basis of our primary research question: *How similar are GitHub projects that share attributes on the documentation side?*.

The findings of our study demonstrate the importance of using data extracted from the documentation side of GitHub projects. Our limited research exposed features not used too frequently in the subject of highlighting similar software. Scenarios in which Readme pages and Wiki files are compared connected repositories in similarity clusters the most accurately. Sufficiently effective outcomes were achieved in the scenario where all the dimensions (Readme, Wiki, and Comments) were combined, with a slight decrease in performance. One of the identified potential issues was related to the presence of commented chunks of code that could negatively influence the outcomes of our experiments.

We believe that analyzing the similarities in the documentation of different repositories can represent a valuable asset in identifying similar applications or tools in utility or implementation. This can further improve the efficiency of finding repositories representing a role model for a newly conceptualized project, considering that relevant documentation is already present.

In addition, we want to reiterate the identified tools in Section 3. The state-of-the-art tool Repopal proposed the usage of Readme files, in cooperation with the tracking of starred projects by users, to identify similar repositories [2]. In comparison, the tool CrossSim proposed a methodology that relied on the interaction between the users and their code, such as dependencies usage [5]. It can be observed that these studies depended on multiple individual dimensions. We believe that a concrete continuation of our research would be to merge our studied dimensions with additional external user activities that can be obtained through the GitHub repositories. A separate research experiment that can be performed would include the comparison between the performance of the proposed Repopal implementation and an adapted version that considers the additional documentation dimensions discussed in this paper.

However, limitations in our study still need to be considered. Our study relied on modern natural language processing techniques to capture the semantic connotation of words and phrases. A beneficial approach that was considered but not developed and evaluated was using a large language model to allow the usage of a deep learning algorithm.

During experiments, we observed that our similarity scores were affected by the default template text used in Wiki pages. Besides extending our stopwords dictionary, an additional idea would be identifying and removing such sentences before processing the text. This technique could also be used to remove auto-generated comments, while identifying commented chunks of code would be possible by appending additional key terms to the stopwords list. The remaining terms dictionary defined in our heuristics and used for vector weight adjustment did not significantly alter our results. We concluded that its selection of words was unsuitable and required massive improvements. As we established our dataset, we only considered repositories with documentation in English. Therefore, we did not create stopwords dictionaries in foreign languages.

Two of our proposed dimensions, URLs and Licenses, did not produce satisfying results. Nevertheless, we do not consider these dimensions a deadlock but an interesting opportunity to complement further and improve our study, starting from our proposed methodology, which did not perform at the expected level.

Technically, we encountered several issues with the GitHub APIs related to verifying the existence of a Wiki page for a project. Wikis on GitHub are handled as separate repositories, therefore having independent privacy statuses (private or public availability). Often, we encountered public repositories that, in our opinion, implemented private Wikis, while the API confirmed the existence of the Wiki. This forces our algorithm to throw an error instead of skipping this step. The second limitation encountered related to the Wiki pages was concerning their naming conventions (which often contain colons in their names) and the operating system on which our algorithm was executed. This issue is caused by our implementation of first downloading all the Wiki files

and then extracting the data, but it can be mitigated by using a Unix-based operating system instead of Windows.

Moreover, data evaluation was one of our experiments' most difficult processes throughout the study. For example, during our clustering experiment, we could not get accurate results, but rather a tendency of them, which was then evaluated. We believe that more sophisticated evaluation procedures could only benefit our study and improve its credibility. Due to time limitations, a user evaluation methodology was not feasible, but it would have been critical in further validating our results in a longer time stamp.

Threats to validity

Due to the nature of our study, multiple points can be raised as threats to external validity. Our dataset is constituted mainly of Java-based projects. While support to identify comments from different programming languages has been implemented, no extensive practical testing has been performed. However, the risk of completely changing the results obtained is minimal since part of our dataset contained repositories using more than one programming language during our experiments.

A different setup and distribution of the dataset could obtain different results from ours, and therefore, we limited ourselves to evaluating the phenomena observed in the data. We cannot guarantee that the results will completely coincide unless our study is validated using a dataset including a more extensive and diverse set of repositories. However, we believe that by running our own validation experiment, we decreased the potential occurrence risk of this threat.

The internal validity of our empirical study can be represented by the possible bias during the evaluation process, which was manual, during multiple sessions. We tried to reduce the effect of this study by splitting the projects based on their topics and only after directly comparing their goals. Another internal threat could be caused by our own dataset creation process, where we tried to filter out substandard repositories through our selection heuristics, but no code was examined.

The threats of internal validity can be diminished by following two methodologies. Firstly, we can conduct additional evaluation experiments with data extracted from more diverse sources, such as a broader range of programming languages used in repositories. Secondly, we consider that an approach that could enhance the trustworthiness of our observations would consist of a user evaluation process, where a group of individuals asses the repositories pairs marked as similar.

Finally, the research was performed assuming the definition of similarity detailed in Section 4. While this definition is in no way complete, a different perception of it would nullify the experiment and, therefore, our findings.

8 Summary

This paper aimed to evaluate a novel approach to studying the reference of publicly available software by relating them using their available documentation. We gathered our data by extracting text from Readme and Wiki files and developer comments in the source code. The data available in these three dimensions is cleaned of insignificant information and standardized through the usage of natural language processing techniques to retain only the nonderived form of the words. This facilitates gathering each word's importance by using TF-IDF in a vector used to output a similarity score, courtesy of cosine similarity.

Our results showed that connecting repositories based on their documentation represents a valid approach that offers a large window of opportunity to explore this domain further and optimize our findings.

9 Responsible Research

Our study is committed to respecting responsible research practices. All experiments can be replicated by following the provided instructions, and we have sourced our datasets from publicly available repositories. We have ensured that no breach of GitHub's usage terms has occurred while mining these repositories. Additionally, all code used in our study is open source and can be found on both GitHub [15] and on the Zenodo open science repository [16]. During the study period, the algorithm used to extract data and calculate similarity scores was developed to provide unbiased results gathered through means that did not involve learning processes. Throughout the studies, our sole goal was to analyze repositories that deployed different amounts of documentation and not to harm the author or team of developers in any way. Even though all the data extracted during our experiments was publicly available, we ensured that we safely discarded all of it at the conclusion of the study.

By adhering to these practices, we strive to make our findings publicly available and contribute to knowledge advancement in the exploration of the relationship between repositories.

References

[1] M. Woodward, "Octoverse 2022: 10 years of tracking open source," Nov 2022.

[2] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on github," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 13–23, 2017.

[3] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, pp. 544–551, 09 2011.

[4] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for idf," *Journal of Documentation - J DOC*, vol. 60, pp. 503–520, 10 2004.

[5] P. T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio, "An automated approach to assess the similarity of github repositories," *Software Quality Journal*, vol. 28, pp. 595–631, Jun 2020.

[6] M. Auch, M. Weber, P. Mandl, and C. Wolff, "Similarity-based analyses on software applications: A

systematic literature review," *Journal of Systems and Software*, vol. 168, p. 110669, 2020.

[7] A. A. Awan, "What is tokenization? types, use cases, implementation," Sep 2023.

[8] A. S. Gillis, "What is lemmatization?: Definition from techtarget," Mar 2023.

[9] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," *Proceedings - International Conference on Software Engineering*, pp. 364–374, 06 2012.

[10] W. Usino, A. S. Prabuwono, K. H. S. Allehaibi, A. Bramantoro, H. A, and W. Amaldi, "Document similarity detection using k-means and cosine distance," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, 2019.

[11] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, vol. 22, pp. 3219–3253, Dec. 2017.

[12] P. T. Nguyen, J. D. Rocco, R. Rubei, and D. D. Ruscio, "Crosssim: Exploiting mutual relationships to detect similar oss projects," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 388–395, Aug 2018.

[13] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," in *A practical approach to microarray data analysis*, pp. 91–109, Springer, 2003.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[15] A. C. Turcu, "Research documentation similarity." Available at https://github.com/acturcu/research-documentation-similarity.

[16] A. C. Turcu, "acturcu/research-documentation-similarity: v1.0," Jan. 2024. Available at https://zenodo.org/records/10573230.