Analysis and Visualization of Similarities of Software Systems

Daniel Atzberger[*]
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

Robert Schwanhold
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

Christian Warmuth
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

Willy Scheibel
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

Daniel Limberger
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

Jürgen Döllner
Hasso Plattner Institute,
Digital Engineering Faculty,
University of Potsdam, Germany

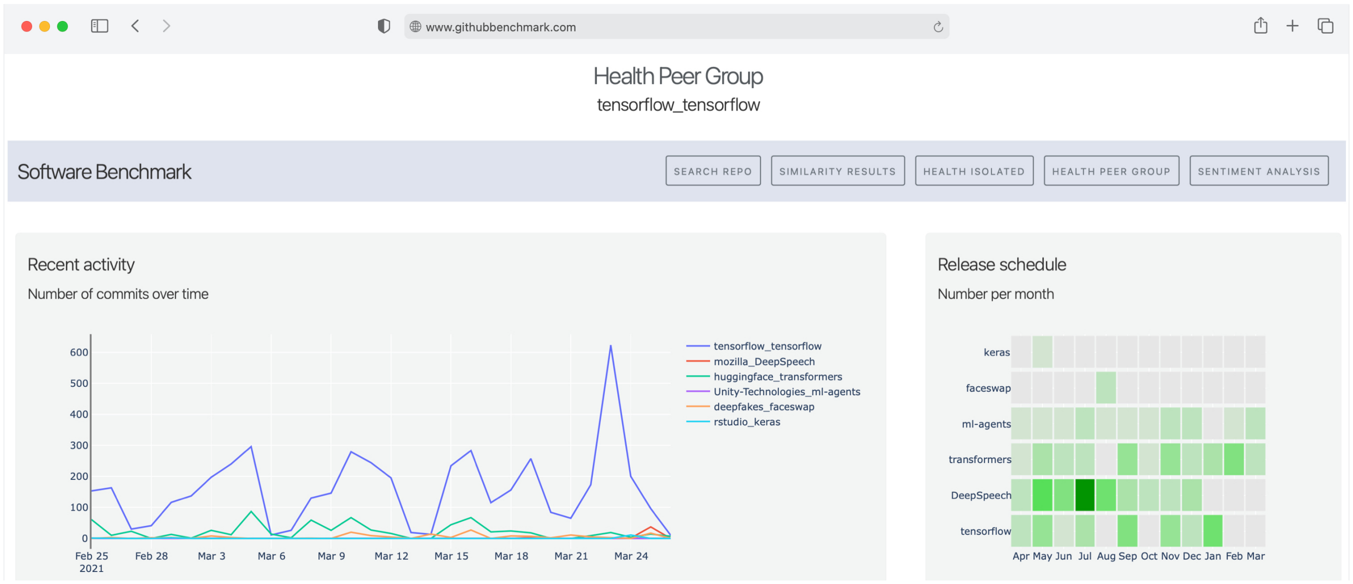


Figure 1: Interface of our peer-group-specific analysis suite for software repositories

ABSTRACT

In 2020, on GitHub, more than 60 million new repositories were created by over 56 million users. This imposes new challenges and opens up new opportunities for the automated analysis and visualization of the software data. When comparing software projects or assessing software projects' health, it is beneficial first to determine repositories similar in content. Therefore, we have developed a tool that allows for the peer-group-based analysis of software repositories from GitHub. The analysis is divided into three parts: similarity search, descriptive analysis, and prescriptive analysis, which we embedded into one website. The similarity search consists of four machine learning models to deliver a combined similarity score. The descriptive analysis allows an isolated view of the repository with multiple metrics and a comparison between peers regarding developer contribution. In the prescriptive analysis, we analyze developer communication to find the most negative user comments/issues to allow targeted support efforts. Our experimental evaluation on over 1000 GitHub repositories for analysis, especially for the similarity search, shows promising results.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; • **Software and its engineering** → **Search-based software engineering**.

KEYWORDS

Software Analytics, Software Visualization, Natural Language Processing, Similar Repositories, Topic Modelling

ACM Reference Format:
Daniel Atzberger, Robert Schwanhold, Christian Warmuth, Willy Scheibel, Daniel Limberger, and Jürgen Döllner. 2021. Analysis and Visualization of Similarities of Software Systems. In *VINCI 2021: International Symposium on Visual Information 6–8 September 2021, Potsdam, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.xxxxxxx

[*] {daniel.atzberger, willy.scheibel, daniel.limberger, juergen.doellner}@hpi.de, {robert.schwanhold, christian.warmuth}@student.hpi.de

VINCI 2021, 6–8 September 2021, Potsdam, Germany
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.xxxxxxx

1 INTRODUCTION

Software development is a data-intensive field with increasing importance. In the year 2020 alone, users created over 60 million repositories and made more than one billion contributions on GitHub, a popular platform for collaborative software development with over 56 million users[1]. Despite this vast amount of data, engineers and managers often lack the tools and techniques to use these potentially powerful information resources to make decisions, assess or improve their projects [5].

When trying to derive meaningful insights and improvement opportunities, it is common practice to limit the projects to compare against. Different software engineering domains like computer graphics, cloud infrastructure, or machine learning differ substantially in their best-practices, community standards, and content. One way of selecting peers would be manual effort since not all repositories are labeled to categorize them into a distinct domain. Furthermore, selecting relevant peers should be flexible since repositories can change their focus, and projects might be semantically more similar to projects in other software engineering domains than their own. Automatically finding similar repositories by using machine learning (ML), in particular ML models for detection of similarities in text data, can be a way to facilitate this endeavor.

Therefore, this project deals with the peer-group-specific analysis of software systems from the peer group search to descriptive analysis and prescriptive analysis. While a descriptive analysis covers descriptive statistics about software projects in an isolated view or in the peer group, the prescriptive aims at predicting future events or suggestions for improvement based on historical and current data. We first develop an ensemble learning model consisting of four different machine learning models common in Natural Language Processing (NLP) to derive the most similar repositories. In the descriptive analysis, the repositories are compared and visualized, using multiple indicators such as the McCabe cyclomatic complexity (MCCC) or the nesting level per module or file. Furthermore, we compare information such as the number of stars or forks over time in the peer group. We additionally implemented a sentiment analysis on developer communication as part of the prescriptive analysis to focus support efforts on the most critical and most negative user voices. We test this approach empirically on a set of 1000 repositories from GitHub

The remainder of this paper is structured as follows: Section 2 summarizes the related work in this field of similarity search for software projects and peer-group-based software analysis. Section 3 lays the theoretical foundation for the similarity search as well as software analysis. In Section 4, we describe our approach and implementation of our solution. We discuss our results in Section 5. Section 6 summarizes our contribution and outlines future work.

2 RELATED WORK

2.1 Similarity Search

Finding similarities in given text corpora is a prominent field in natural language processing research. Deerwester et al. introduced the Latent Semantic Indexing (LSA /LSI) [7], which uses Singular Value Decomposition on the term-document matrix to find a semantic structure between terms and documents. Thereby, documents can be represented as vectors of factor weights. The similarity of two documents is calculated by applying the cosine similarity to the document vectors.

Blei et al. developed Latent Dirichlet Allocation (LDA) [3], which uses a mixture model to retrieve latent semantic information from documents. Each document is represented as a mixture of topics, whereas each topic is represented as a mixture of words. The topic distributions of different documents are compared with the cosine similarity or the Jensen-Shannon divergence. LSI and LDA's primary use case is to retrieve information from natural text documents. However, research has shown that these techniques can also be applied to software repositories [6]. Linstead et al. [21] used LDA in the Sourcerer framework to detect similarities between source code files. Additionally, the relationship between authors and topics is captured with author-topic modeling (AT), an LDA extension.

Instead of the source code itself, other information, such as the readme file, can be used as the basis for similarity detection. Sharma et al. used heuristics to extract the important paragraphs from the readme and then applied LDA-GA to it [24]. LDA-GA is a combination of LDA, and Genetic Algorithms [23].

The detection of similar source code is not limited to LSI or LDA. Zhang et al. [28] developed the RepoPal algorithms, which combines three heuristics: the intersection of words from two different readmes; the common starred repositories of users; and the repositories starred by the same user in a short period of time. Di Sipio et al. [9] combined the term frequency-inverse document frequency (tf-idf) with a Naïve Bayesian classifier, a probabilistic machine learning network for searching similar repositories. Sheneamer et al. [25] used Java bytecode dependency graphs (BDG), program dependency graphs (PDG), and abstract syntax trees (AST) to capture semantics and structure. Based on this, features are extracted and compared with machine learning to find similarities in source code.

2.2 Analysis of software projects

Software quality metrics aim to assess software quality and measure how maintainable a piece of software is. Common metrics include lines of code, McCabe's cyclomatic complexity (MCCC), and the number of functions [1]. By visualizing these metrics, the user gains insight into the data, such as how the MCCC differs between different modules. [4]. Additionally, it has been shown that visualization reduces the time required for data interpretation compared to reading raw data [22].

Multiple projects have been developed to visualize different aspects of software engineering. Sourcerer[2] is a project where developers can analyze themselves by analyzing the repositories they work on. The analysis includes data from GitHub, GitLab, and Bitbucket. Visual presentation of the user's development experience and a summary of all associated repositories are some of the main features. Seerene[3] developed a software analytics and process mining technology, called Digital Boardroom, that analyzes and visualizes the software development processes of a company. It reveals weaknesses and gives actionable insights.

[1] octoverse.github.com/, github.com/about

[2] sourcerer.io/

[3] www.seerene.com/

However, these projects have a different focus. Repositories are either analyzed within the context of a company or a specific developer. Therefore, the peer group aspect, which allows further comparisons between the repositories, is currently not covered.

3 SIMILARITY SEARCH

Programming languages and source code share many statistical properties with natural language. Similar to natural languages, source code is repetitive and can be modeled by statistical language models [11]. Hence, Natural Language Processing (NLP) approaches, such as LSI or LDA, also apply to source code and can derive semantic similarities between software projects.

3.1 Data Foundation and Preprocessing

Our analysis of similar repositories is based on two different data sources.

Source Code: The primary data source is the source code of the repositories. Similar repositories, in the sense of semantics, implement similar functionality. The names of variables or functions do not affect the execution of a program. However, they are crucial to the readability for developers [15]. Hence, the names of identifiers in practice often represent the underlying semantic. Therefore, the terms used in the source code overlap in similar repositories to some degree.

Readme: The secondary data source are the readmes of software repositories. Most readmes contain a section describing the content of the repository. These descriptions can be used to compare repositories. However, the rest of the readme, such as the installation setup or the license, contains little semantic information. Therefore, extracting the relevant sections of a readme becomes a key challenge. First, we divide the entire readme into different sections using empty lines as separators. Our goal is to select a subset of sections that describe the repository well. Sections at the beginning of the readme are prioritized. Based on heuristics, we decide whether a section is selected or not. For example, each selected section must include at least five words and consist of more than 80% of alphanumeric characters.

Preprocessing steps for readmes and source code: After this filtering step for readmes, we perform preprocessing on readmes and source code. The preprocessing of text data for machine learning tasks is a crucial step to improve the quality of the results [16]. In natural texts, words are split by whitespaces. However, in source code, multiple words are often combined to form identifiers of variables or methods. In this case, camel-case or snake-case notation need to be considered. Therefore, we also split words at underscores or capital letters inside words. Stopwords can be removed because they do not contain semantic information. Since most software projects are written in English, we only remove the stopwords of the English language. Additionally we also remove reserved terms that are programming language-specific, such as *switch* and *return*.

In the next step, we are applying the text to a bigram model. This replaces adjacent tokens with a single token if they form a bigram collocation. Every word can appear in different inflected forms. For example, a verb might be written in different tenses, a noun can be either singular or plural, and an adjective can also be in the comparative or superlative form. The lemmatization reduces all inflections of a word by grouping them into one form of the word. We use the python module *spacy* for this purpose [12].

Depending on the model, we either use the absolute frequency of a word or the term-frequency inverse-document-frequency (tf-idf) as numerical representations. Tf-idf is defined as follows

$$w_{t,R_1} = (1 + log(tf_{t,R})) \times log\left(\frac{|C|}{df_t}\right) \quad (1)$$

The tf-idf value reflects how important a word (w) is for a document (R) in a corpus (C). The value increases proportionally to the number of word-occurrences in the document and is balanced by the number of documents in the corpus containing the particular word. Documents are represented as vectors, containing the tf-idf values for each word in the document [28].

3.2 Similarity Metrics

Latent Semantic Indexing: The idea of LSI is that a text has underlying latent dimensions. Singular Value Decomposition (SVD) expresses the term-document matrix as the product of three matrices ($M = U\Sigma V^*$) and thereby discovers the singular values. Σ represents the diagonal matrix with the singular values, whereas U and V represent the left and right singular vectors. [7]. The singular values order the dimensions according to their significance. Hence, less significant dimensions can be ignored without losing the underlying semantic. The parameter k specifies the number of relevant dimensions. Therefore, the parameter k also defines the length of the vectors of factor weights representing the documents.

For LSI, the repositories' similarity is calculated by applying the cosine similarity to the vectors representing the documents, thereby calculating the cosine of the angle between the two vectors. The range of values of the cosine function is [-1; 1]. Values close to 1 mean *similar* and values close to -1 mean *not similar*. A value of 0 means that the vectors are *orthogonal*.

Latent Dirichlet Allocation: Latent Dirichlet Allocation is a generative probabilistic topic model to retrieve semantic from documents [3]. Documents are modeled as a probability distribution over topics, whereas topics are modeled as a probability distribution over words. LDA is based on the assumption of exchangeability, which uses a *bag-of-words* semantic. When using the bag-of-words approach, the order of words inside a document and the order of documents inside the corpus are neglected. Given a document corpus of size D, where the number of words inside a document d is denoted as N_d, the goal is to infer the per-word topic assignment $z_{d,n}$, the per-document topic proportions θ_d, and the per-corpus topic distributions β_k (with $d \in \{1, 2, ..., D\}$, $n \in \{1, 2, ..., N_d\}$, $k \in \{1, 2, ..., K\}$). These variables are latent, meaning that they are not observed. Instead, they are inferred with an approximate posterior inference algorithm, for example, using Gibbs sampling. [3]

There are multiple parameters for model tuning:

- K: number of topics corresponds to the number of dimensions of the Dirichlet distribution.
- α: is a parameter that can tune the shape of the Dirichlet distribution of θ. This parameter can be used to control the number of topics that are present in a document. Setting α close to 0 would result in a single topic per document.
- η: is a parameter that can tune the shape of the Dirichlet distribution. This parameter controls the number of words

that are present in a topic. Setting η close to 0 would result in few dominant words per topic.

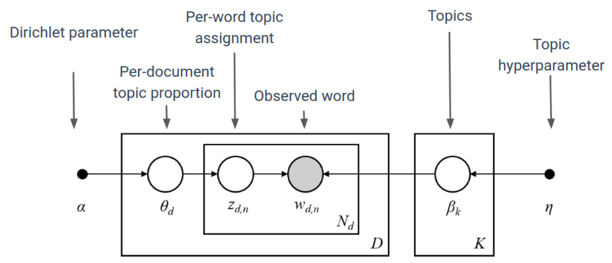


Figure 2: Graphical model representation of LDA

For computing the similarity in the context of LDA, we apply the Jensen-Shannon divergence to the vectors representing the repositories. This divergence only works for probability distributions and therefore does not apply to LSI. It is based on the Kullback-Leibler divergence but additionally satisfies symmetry. The value range for two probability distributions is [0; 1], where 0 means that the probability distributions are the same [20, 26].

Repopal Similarity Score: The Relevance Score compares how much the words of two documents match. Instead of absolute word frequencies, the term-frequency inverse-document-frequency is used in the calculation. This prevents the result from being distorted by documents of different lengths [28]. The similarity of two documents can now be calculated using cosine similarity.

$$\text{Relevance}_r(R_1, R_2) = \frac{\sum_{t \in R_1 \cap R_2} w_{t,R_1} \times w_{t,R_2}}{\sqrt{\sum_{t \in R_1} w_{t,R_1}^2} \times \sqrt{\sum_{t \in R_2} w_{t,R_2}^2}} \quad (2)$$

Penalized NMF: Semi-supervised clustering is a clustering technique that makes use of the limited guidance by some supervisory information. The search for semantically similar repositories is an unsupervised approach, while in this case, supervisory information can be obtained from the user in the form of must-link and cannot-link constraints. Fei et al. define must-link and cannot-link constraints as follows: For must-link constraints, “the two data points must belong to the same class” while for cannot-link constraints “the two data points cannot belong to the same class” [27]. The user can provide feedback by removing or adding repositories to the initial recommendation of peers to a reference repository.

Nonnegative matrix factorization (NMF) is a dimensionality reduction and clustering method that tries to factorize a nonnegative matrix into two low-rank nonnegative matrices. This method is commonly used for clustering document data and as a topic modeling method [17]. Traditional NMF tries to approximate the nonnegative matrix V by the product of two lower-rank matrices W and H such that $V \approx WH$ using the following objective function:

$$\min_{W,H \geq 0} \|V - WH\|^2 \quad (3)$$

V has the dimensions $m \times n$, W has the dimensions $m \times r$ and V has the dimensions $r \times n$ while r denotes the number of lower dimensions. m in our application denotes the number of repositories, and n the number of feature words considered with bag-of-words or tf-idf representation for all repositories [17].

Penalized NMF introduces a constraint matrix X. A possible constraints matrix is displayed in the following, which encodes in our case a cannot-link constraint between the repositories (2,3) and (3,2) and a must-link constraint between (3,5) and (5,3) considering an index starting with one.

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} \quad (4)$$

The objective function of NMF can be penalized with the matrix X as follows to enforce a shift towards or away from each other in lower-dimensional space during clustering:

$$\min_{W,H \geq 0} \|V - WH\|^2 + tr(W^T XW) \quad (5)$$

To get the similarity between two repositories after the penalized clustering, one can calculate the distance in the lower-dimensional space using the cosine similarity or Euclidean distance [17, 19, 27].

3.3 Combination of all similiarity metrics

Instead of using a single similarity metric, we are combining the different metrics into an ensemble model. Even though the user’s feedback improved the penalized clustering results, this metric’s overall quality remained inferior compared to the other similarity metrics. This is attributed to the bad initial clustering. Repopal similarity score is a relatively simple similarity metric as it only sums word co-occurrences and does not provide good results compared to the presented topic models. Due to the poor quality of results from the penalized NMF and the Repopal score, we only consider the topic models:

- LSI on the readme
- LSI on source code
- LDA on the readme
- LDA on source code

The Jensen-Shannon divergence and cosine similarity have different result ranges. Therefore, we map all similarity score on the interval [0, 1], where values close to 1 mean *similar* and values close to 0 mean *not similar.* Therefore, the correct interpretation of this value is that of a similarity score, not a distance measure. The single scale allows the unified handling of the similarity scores from the different models. The overall score is calculated by taking the average of the individual scores. Optionally, a weighted average could be used, where we can prioritize a specific model.

4 PROTOTYPE IMPLEMENTATION

4.1 Technical Setup

The results of our peer-group-based analysis are visualized on a website. For the website, we use Flask as a web application framework for the python language [10]. It has a low entry barrier but is still very versatile, which allows for quick implementation of prototypes. We are using Dash, a framework that builds on top of Flask [4]. This allows us to create interactive visualizations with Plotly [14] in python and display them on the website.

On the homepage of our website, the user can choose a repository of interest. The rest of the website is structured into four dashboards: similarity results, health isolated, health peer group, and sentiment analysis. The first dashboard presents the results of the similarity analysis. Both dashboards health isolated and health peer group form together with the descriptive analysis. The sentiment analysis represents the prescriptive analysis. The hosting of the website and all the time measurements in this paper are performed on a server (see Table 1) with the following specifications.

Table 1: Overview of the hardware configuration of the server for time measurements

Description	Component
Processor	AMD Ryzen 5 3600
Number of Cores	6
Size of Main Memory	64 GB DDR4
Hard drive	2 x 512 GB NVMe SSD

4.2 Similarity Analysis

Training topic models requires a large corpus of documents. The corpus needs to be diverse to cover a broad spectrum of repositories. Simultaneously, for each repository, there must be repositories semantically close to it to get meaningful results. We used a ranking of the most popular repositories on GitHub, grouped by category [18], as the baseline for creating our corpus. The table contains over 2700 entries for repositories. The categories include the top repositories for many programming languages and the "top-100-stars" and "top-100-forks" overall. We only used the categories of the common programming languages and "top-100-stars" and "top-100-forks". We discovered that some of the very popular repositories only contain little source code if any. Therefore, we additionally filter out repositories with less than ten source code files. As a result, our corpus consists of over 1000 repositories. For the local analysis process, we clone the repositories to the local file system. The size of all repositories adds up to roughly 250GB. Training the ensemble model on the corpus takes 3 hours. This grows linearly with the number of repositories. We are limiting the model to use at most 100 source code files per repository for computational reasons. The model can be persisted on disk and loaded again at some point in time without recomputing the model. Calculating similar repositories for all repositories from our corpus requires a pairwise similarity computation. This takes roughly 50 hours and grows quadratically with the number of repositories. Therefore, given a single repository, it takes about 3 minutes to compute its peer group. We store the list of similar repositories, the average similarity score, and the different models' scores in a CSV-file for later processes.

Under the assumption that a repository's content does not change drastically in a short time, this process does not need to be updated very frequently. This assumption may not hold for young repositories. Since our corpus consists of the top repositories of the different categories, we can assume that their content and it's overall semantic does not change drastically in a short period. Rerunning the similarity computation once a month seems sufficient.

Similarity Results Dashboard: This dashboard visualizes the results of the similarity calculation. For each of the similar repositories, we list the average score and the individual scores. In the dashboard, we show the five most similar repositories according to our similarity metric introduced in Section 3. The resulting similarity score is broken down into the individual metrics for each of the peers as shown in Figure 3.

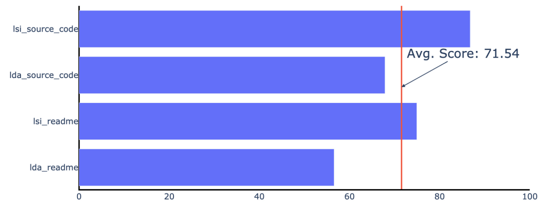


Figure 3: Breakdown of similarity results for one of the peers to TensorFlow

A force-directed k-nearest neighbor graph shows the connections of the different repositories. An example of this force-directed graph is shown in Figure 4. The graph is limited to indirect neighbor nodes from the selected repository with a distance smaller or equal to two for readability. Two nodes in this graph n_1, n_2 are connected, if n_2 belongs to the top-5 similar repositories of n_1 (or vice versa). The edge attraction of the force-directed graph is dependent on the similarity score introduced in Section 3.

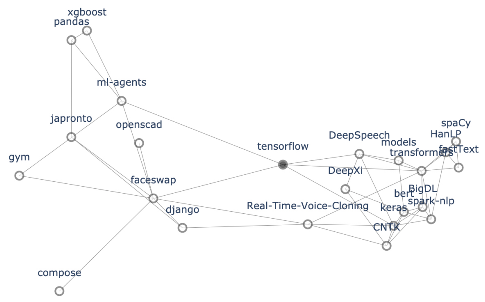


Figure 4: Force-directed k-nearest neighbor graph for TensorFlow

Furthermore, basic metadata about the corpus is displayed. This includes, for example, the number of total repositories and the distribution of languages in the corpus.

Given this overview of the similarity results for the suggested peer group, the user can add or remove repositories from the peer group. We store the user's feedback to improve our future suggestion by incorporating the feedback into the ensemble model. The peer group that the user selects here will influence the result on the "Health Peer Group"-dashboard.

[4] plotly.com/dash/

4.3 Descriptive Analysis

Since software projects' descriptive analysis is a large subject in software analytics, we rely on existing solutions. We use the local analyzer from Seerene[5] to analyze all local repositories. We store the result of the local analysis in three files:

- `history.csv` contains an entry for each change, alongside meta information such as the date, commit message and commit-ID
- `items_meta.json` groups all source code files by language
- `metrics.csv` contains various metrics for each source code file. For example, McCabe's cyclomatic complexity, number of TODOs, lines wider than 140 characters, hacks, or the nesting level.

We aggregated the number of TODOs, source code lines, comments, and commented-out code lines for all repository files. The analysis of all 1000 repositories takes roughly an hour, and the results are persisted on disk. This process should be updated at regular intervals to keep the results up to date. This also requires checking out the newest master branch of the local repositories to get the latest commits. A suitable interval would be every few days.

Health Isolated Dashboard: This dashboard facilitates a descriptive analysis of the selected repository without comparing it to the peer group. The main component of this dashboard is an interactive treemap, as shown in Figure 5. Each code file represents a node in the treemap. The color is used to map one of multiple metrics to the file, while the area size per file corresponds to the lines of code.

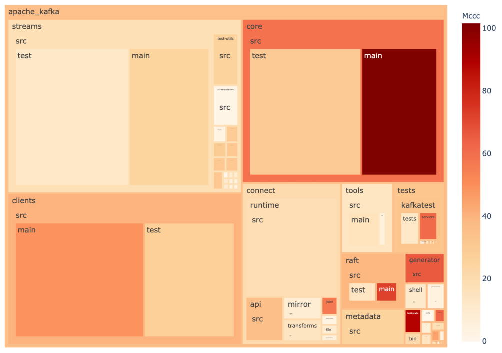


Figure 5: Treemap for isolated repository analysis using the McCabe cyclomatic complexity

The user can switch between the following metrics: complexity, number of TODOs, number of lines wider than 140 characters, number of hacks, McCabe's cyclomatic complexity, and nesting level. This dashboard uses the data resulting from the local analysis.

Health Peer Group Dashboard: This dashboard compares important statistics between the different repositories. A heatmap visualizes the number of releases per month for the different repositories in the peer group (see Figure 6). More releases per month are expressed with a darker color. This enables the user to get an overview of which repositories are maintained and released regularly.

[5] www.seerene.com/

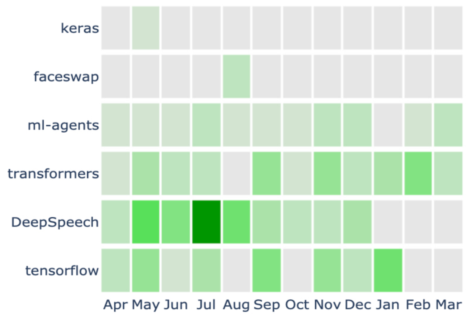


Figure 6: Release schedule of TensorFlow and its peer group

Key statistics, such as the number of commits per day, stars, forks, open issues, or open PRs over time are presented as a multi-line chart (see Figure 7).

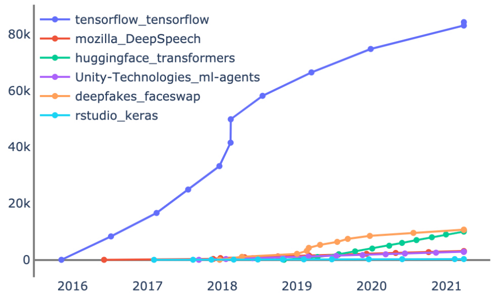


Figure 7: Forks over time of TensorFlow and its peer group

A stacked bar chart, as shown in Figure 8 is used to compare the distribution line types (source code, comments, commented-out source code) between the peer group's repositories. This also gives an overview of the selected repository's relative sizes compared to the peer group.

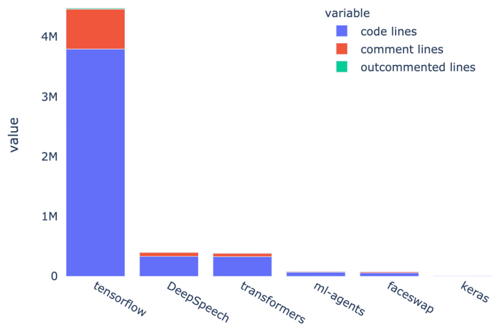


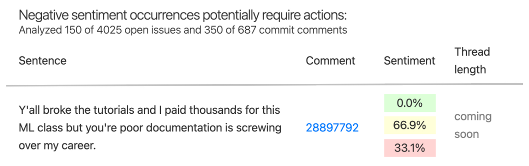
Figure 8: Stacked bar chart for line type distribution for TensorFlow and its peer group

All of the statistics, except the commits, are queried live via the GitHub API, taking up to 30 seconds to load. Loading symbols indicate that data is loading in the background.

4.4 Prescriptive Analysis

The prescriptive analysis idea is to give pointers to issues where an owner of a repository might need to intervene because we assume that a focus on the most negative user voices is beneficial. Negative user voices can either point to problems associated with a project the community is frustrated about and point towards heated off-topic conversations that need moderation or interventions.

Sentiment Analysis Dashboard: Sentiment analysis is an NLP technique for classifying sentences' sentiment into positive, neutral, or negative classes. For that purpose, we use VADER (Valence Aware Dictionary and sEntiment Reasoner), a pre-trained sentiment classifier trained on social media texts [13]. Comments from open issues or commits are extracted live from the GitHub API as the basis for the sentiment analysis. The analysis calculates a sentiment distribution (positive, neutral, negative) for each sentence. Negative comments may indicate that, for example, functionality is broken (see Figure 9).



Negative sentiment occurrences potentially require actions:
Analyzed 150 of 4025 open issues and 350 of 687 commit comments

Sentence	Comment	Sentiment	Thread length
Y'all broke the tutorials and I paid thousands for this ML class but you're poor documentation is screwing over my career.	28897792	0.0% 66.9% 33.1%	coming soon

Figure 9: Example of negative comment in the TensorFlow repository

For performance reasons, the number of open issues is limited to 150, and the number of commit comments is limited to 350, which takes up to 40 seconds combined.

5 DISCUSSION

This project's goal was to cover all three aspects of the peer-group-based analysis (similarity search, descriptive analysis, prescriptive analysis) while focusing on the similarity search. We have achieved promising results for the similarity search and incorporated all parts into a website.

Evaluation of Similarity Results: Since the similarity search is rooted in unsupervised machine learning, there are no objective measures that qualitatively evaluated the calculated similar repositories' correctness. Despite subjectively evaluating the similarity search, no user research study has been conducted to evaluate the results objectively. We, therefore, suggest the execution of a user study. As a precondition to the user study, the participants need to be familiar with GitHub and popular open-source repositories. Then, three variations of user studies can be performed. In one variation, users rate the similarity search's output on a scale from 1 to 5; in a second variant, the user searches independently for the five most common repositories to a reference repository. Then we compare results to the results given by the similarity search. The third variant would be "intrusion research", similar to the approach "topic intrusion" for topic models [2]. In this case, the user is presented with a peer group to a reference repository generated by the similarity search and an additional "intruder" repository. The user has to correctly guess the "true intruder" to measure how closely related the calculated ones are.

Additional similarity metrics: One could use pre-trained embeddings or transformer models such as Bidirectional Encoder Representations from Transformers (BERT) developed by Google[8] or Word Mover Distances. In conjunction with a user study, multiple other models can be incorporated into the similarity score if they prove to improve the overall results of the similarity search.

Sentiment Analysis Model: In the current state, we use a pre-trained sentiment analysis model (Vader). The dataset used to train this model is not specific to the field of software engineering. Therefore, messages about errors that are, in fact, neutral are falsely categorized as "negative" due to the existence of marking words as "error", "wrong", or "not working". If we would label enough developer communication data, we could train a sentiment analysis model specific to developer communication. Here, feedback from the user if an issue or comment is indeed negative can also be helpful to further tune the sentiment analysis in productive use.

Loading times: Currently, the analysis is based on a predefined set of repositories without the ability to upload or use private or company-internal software repositories. When adding a new repository (private or public), the local analysis pipeline needs to be recomputed for the new repository only, which takes a few seconds. However, the re-calculation or calculation of similarities for new repositories turns out not to be possible in seconds. Calculation of the most similar peers for a new repository is possible without recomputing the ensemble model. However, this repository can only be seen in the peer group for other repositories if the complete ensemble model is retrained again from scratch.

Limitation of the GitHub API: For most visualizations, we rely on the GitHub API. For that purpose, we used the python module `PyGithub`[6]. However, we experienced limitations and inconsistencies with the API:

- The number of contributors from the API does not match the number available on the GitHub website. Possible reasons for this might be that a single user is associated with more email addresses, which is handled differently in the API.[7]
- Pagination for stars is limited to 40.000 entries. If a repository has more than 40.000 stars, only the metadata of the first 40.000 stars can be accessed. This metadata is important to plot the number of stars over time. We mitigated the problem partially, in this case, by interpolating the line plot with the current number of stars.

Furthermore, the GitHub API limits the number of requests to the REST-API (v3) and the GraphQL API (v4) to 5.000 per hour.[8] One way to overcome this problem would be to introduce a token-donation along with user management. This way, each user can only access the resources if he donates tokens.

Private Projects and Public Website: Currently, only publicly available repositories can be incorporated in the similarity search and further analysis. Without user management, this is not possible as the private repositories should only be available to the respective user. Furthermore, for cloning private repositories directly from GitHub or allowing file upload, the model pipeline needs to be changed considerably to separate private or company-internal user

[6] pygithub.readthedocs.io/en/latest/introduction.html
[7] github.community/t/api-github-list-contributors/13520
[8] docs.github.com/en/graphql/overview/resource-limitations

content. Then, multiple models need to be trained to separate private and public repositories for the peer group recommendations. When we implement user management and allow private repo uploads and implement a token-donation system, then publishing the website would be a good next step.

Descriptive Analysis & Prescriptive Analysis: We presented some ideas for the descriptive analysis and the prescriptive analysis; however, there is still much room for further ideas or use cases. Despite using new additional machine learning models for the similarity search, one could also enable the similarity search for different stages/times in the repository evolution. One possible new insight could be if a repository is more similar to an old state than to the current state of another repository. The descriptive analysis could be extended to compare different repositories based on their complexity or setup. Furthermore, additional information about the test coverage and status of the tests and deployment can indicate software projects' health. Moreover, information about the frequency of changes or the age of contents for different files or modules can serve as additional value. The prescriptive analysis could make more sense of temporal information, such as the shift of topics over time. If the result of the similarity search leads to the result that the repository is more similar to an older stage of a different repository, prescriptive analysis as to what has changed can serve as recommendations for future improvements.

6 CONCLUSION & FUTURE WORK

While there exist ways to derive similar software projects, there is no holistic solution for peer-group-based analysis of software systems. This paper describes our efforts to provide a peer-group-based analysis from the search of similar repositories to descriptive and prescriptive analysis, all bundled in one website.

While our work only constitutes the cornerstone, good results for the derived peer groups and existing and new approaches for the descriptive and prescriptive analysis were laid out, which can be a baseline for further research in peer-group-specific analysis of software repositories.

Future research efforts in this field should study additional similarity models as part of the ensemble model for the peer group search. The similarity search result should also be evaluated in a user study to quantify the results, evaluating whether the new models as part of the ensemble model improve the overall result. A publicly available website with a feature to upload a private or company internal repository and a token-donation-system can further expand this project's target audience.

ACKNOWLEDGEMENTS

This work is part of the “Software-DNA” project, which is funded by the European Regional Development Fund (ERDF – or EFRE in German) and the State of Brandenburg (ILB). This work is part of the KMU project “KnowhowAnalyzer” (Förderkennzeichen 01IS20088B), which is funded by the German Ministry for Education and Research (Bundesministerium für Bildung und Forschung).

REFERENCES

[1] Dur Abuzaid and Scott Titang. 2015. The Visualization of Software Quality Metrics - A Systematic Literature Review. http://hdl.handle.net/2077/38848

[2] Shraey Bhatia, Jey Han Lau, and Timothy Baldwin. 2018. Topic Intrusion for Automatic Topic Model Evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 844–849. https://doi.org/10.18653/v1/D18-1098

[3] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022. https://doi.org/10.1162/jmlr.2003.3.4-5.993

[4] Ken W Brodlie, Lesley Ann Carpenter, Rae A Earnshaw, Julian R Gallop, Roger J Hubbold, Anne M Mumford, Chris D Osland, and Peter Quarendon. 2012. *Scientific visualization: techniques and applications*. Springer Science & Business Media.

[5] R. P. L. Buse and T. Zimmermann. 2012. Information needs for software development analytics. In *2012 34th International Conference on Software Engineering (ICSE)*. 987–996. https://doi.org/10.1109/ICSE.2012.6227122

[6] Tse-Hsun Chen, Stephen W Thomas, and Ahmed E Hassan. 2016. A survey on the use of topic models when mining software repositories. *Empirical Software Engineering* 21, 5 (2016), 1843–1919.

[7] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R.A. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41* (1990), 391–407.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[9] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T. Nguyen. 2020. A Multinomial Naïve Bayesian (MNB) Network to Automatically Recommend Topics for GitHub Repositories. In *Proceedings of the Evaluation and Assessment in Software Engineering* (Trondheim, Norway) *(EASE '20)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/3383219.3383227

[10] Miguel Grinberg. 2018. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.".

[11] Abram Hindle, Earl T Barr, Mark Gabel, Zhendong Su, and Premkumar Devanbu. 2016. On the naturalness of software. *Commun. ACM* 59, 5 (2016), 122–131.

[12] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. https://doi.org/10.5281/zenodo.1212303

[13] C.J. Hutto and Eric Gilbert. 2015. VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014.*

[14] Plotly Technologies Inc. 2015. *Collaborative data science.* Montreal, QC. https://plot.ly

[15] Alan Jaffe, Jeremy Lacomis, Edward J Schwartz, Claire Le Goues, and Bogdan Vasilescu. 2018. Meaningful variable names for decompiled code: A machine translation approach. In *Proceedings of the 26th Conference on Program Comprehension*. 20–30.

[16] Vaishali Kalra and Rashmi Aggarwal. 2017. Importance of Text Data Preprocessing & Implementation in RapidMiner.. In *ICITKM*. 71–75.

[17] Da Kuang, Jaegul Choo, and Haesun Park. 2015. *Nonnegative Matrix Factorization for Interactive Topic Modeling and Document Clustering*. Springer International Publishing, Cham, 215–243. https://doi.org/10.1007/978-3-319-09259-1_7

[18] Evan Li. 2021. Github-Ranking. https://github.com/EvanLi/Github-Ranking/blob/master/Data/github-ranking-2021-02-07.csv. [Online; accessed 07-March-2021].

[19] T. Li, C. Ding, and M. I. Jordan. 2007. Solving Consensus and Semi-supervised Clustering Problems Using Nonnegative Matrix Factorization. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 577–582. https://doi.org/10.1109/ICDM.2007.98

[20] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151.

[21] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery* 18, 2 (01 Apr 2009), 300–336. https://doi.org/10.1007/s10618-008-0118-x

[22] Chen M., Floridi L., and Borgo R. 2014. What Is Visualization Really For?. In *The Philosophy of Information Quality*, Vol. 358. https://doi.org/10.1007/978-3-319-07121-3_5

[23] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimilano Di Penta, Denys Poshynanyk, and Andrea De Lucia. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 522–531.

[24] Abhishek Sharma, Ferdian Thung, Pavneet Singh Kochhar, Agus Sulistya, and David Lo. 2017. Cataloging GitHub Repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (Karlskrona, Sweden) *(EASE'17)*. Association for Computing Machinery, New York, NY, USA, 314–319. https://doi.org/10.1145/3084226.3084287

[25] Abdullah Sheneamer, Swarup Roy, and Jugal Kalita. 2018. A Detection Framework for Semantic Code Clones and Obfuscated Code. *Expert Systems with Applications* 97 (05 2018). https://doi.org/10.1016/j.eswa.2017.12.040

[26] Zhou Tong and Haiyi Zhang. 2016. A text mining research based on LDA topic modelling. In *International Conference on Computer Science, Engineering and*

Information Technology. 201–210.
[27] Fei Wang, Tao Li, and Changshui Zhang. 2008. Semi-Supervised Clustering via Matrix Factorization. 1–12. https://doi.org/10.1137/1.9781611972788.1
[28] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, and J. Sun. 2017. Detecting similar repositories on GitHub. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 13–23. https://doi.org/10.1109/SANER.2017.7884605