Assessing Software Project Similarity[☆,☆☆]

Mattias Blaim[1], Malik Faizan Ahmed[2], Manuel Raffel[3], Roman Walser[4]

Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

Abstract

Although many software project repositories on GitHub contain applications fulfilling a similar purpose, there is no functionality available to actually find similar projects. Learning from similar projects could, however, increase the overall quality of ongoing projects as code can probably be reused and mistakes can be avoided. This paper describes a first approach on measuring similarity of two projects in order to classify them as similar, rather similar, rather different or different. Results show that it is, with limitations, possible to use similarity measures for determining whether two software projects are similar or not.

Keywords: Project Similarity, Similarity Measures, Finding Similar Software Projects, GitHub Repositories, Project Success, Classification, Prediction Model

[☆] This document is a collaborative effort. The work plan and division of labor regarding this paper and the overall project can be found in the Appendix, Figure 5.

[☆☆] This seminar paper is subject in the course Business Process Analytics. The purpose of the paper is to get familiar with the recent research discussion in this field and to be able to identify business scenarios in which complex event processing can be applied.

[1] `mattias.blaim@s.wu.ac.at`, Student ID: 1050802

[2] `malik.faizan.ahmed@s.wu.ac.at`, Student ID: 1351069

[3] `manuel.raffel@s.wu.ac.at`, Student ID: 0850648

[4] `roman.walser@s.wu.ac.at`, Student ID: 1152149

1. Introduction

Over recent years, recommendation systems have become object of increased academic attention. Mainly starting in the area of product recommendation in online retail [1], scientific and practical progress led to the development of more sophisticated and context-aware recommendation systems [2]. Even if recommendation systems in the context of software development already exist, they are still at an early stage [3]. This circumstance is also reflected by the fact that there still is no recommendation system implemented in GitHub (one of the world's largest collaborative coding platforms).

However, recommendation of similar projects in software development might not only help to reuse code fragments, but also to learn from other projects overall approaches and procedures. Insights into comparable software projects that have already been implemented successfully could help to significantly reduce the error rate of ongoing projects. Hence, to enable learning from comparable projects, a precise and accurate measurement for assessing project similarity is required. Therefore, the aim of this paper is to find an appropriate set of measured values to determine the degree of similarity between two software projects. The similar projects then could serve as a source of information for increasing the overall quality of an ongoing project. Accordingly, the research question of this paper is the following: "Which similarity measures are suitable to assess similarity of software projects published on GitHub?".

To answer the research question, this paper is structured as follows: Section 2 starts with a theoretical introduction into the topic of measuring project similarity and presents an overview of relevant literature. It also includes background information about GitHub and GHTorrent, the data sources used throughout the paper, as well as the data that is available from them. A set of possible similarity measures that can be calculated using this available data is proposed in Section 3. Subsequently, the actual method of gathering the data and calculating the similarity measures as well as the statistical analysis and their result is discussed in Section 4. The paper closes with a conclusion about the results and consequential contributions and limitations of this research, followed by an outline of possible future work (see Section 5).

2. Background

This section aims to present the background of the conducted research. An overview of research previously done in this field is given and the high relevancy of this topic is pointed out. This also includes background information about GitHub and the GHTorrent project that served as data source for the performed calculations.

2.1. Theoretical Foundation

Already since the 1960s, researchers have been trying to find out which factors determine project success [4]. Following Belassi and Tukel [5], typical success factors can be categorized into the following groups:

- Abilities of the project team members (e.g. technical background and skills, commitment, communication sills, troubleshooting ability)
- Abilities of the project manager (e.g. ability to delegate, trade-off, coordinate, overall competence, perception of role and responsibilities)

- Factors related to the project itself (e.g. size and value, uniqueness of the project activities, urgency, density of the project, life cycle)
- Factors related to the organization (e.g. top and functional management' support, project organizational structure and behavior)
- Factors related to the environment (e.g. technological environment, nature, competitors, sub-contractors, social environment)

Despite the broad knowledge researchers could gain in the field of (successful) project management, a considerable part of software projects still fails [6]. This might be due to the fact that some of the factors mentioned above can not or only hardly be influenced by the responsible project team members. Whereas the project members or the project manager are rather easily modifiable, environment is often given. To give an example, the involved stakeholders will usually stay the same. The more important it is to optimize the factors that can be influenced and to prevent unnecessary mistakes. One way to do so could be to find similar projects as a reference to learn from. But also costs can be reduced if code fragments of similar projects are reusable for an ongoing project.

Also in the context of software engineering, projects have been compared for a long time. However, these comparisons are often limited to effort and cost estimation (e.g. [7], [8]). Azzeh et al. [9] present a software project similarity measurement based on fuzzy C-means. Again, with the overall aim of improving effort estimation, they suggested the following seven effort driving factors for their statistical comparison: adjusted function points, maximum team size, productivity, development types, business type, application types, organization types. The strength of the proposed fuzzy C-means approach was that numerical and categorical values could be combined. However, this information is not always publicly available and therefore has only limited applicability to the topic at hand.

Schmidt et al. [10] investigated project success from another aspect, namely the minimization of risk. This success risk includes similar factors to those listed in Belassi and Tukel [5], but additionally points out the importance of uncertainty as a significant risk for project success. Here again, finding similar projects and learning from experiences made in other projects could clearly help to reduce uncertainty and therefore increase the overall success of an ongoing project.

To sum up, several elaborated sets of critical success factors are already available in research. However, there seems to be a lack of results regarding comparisons applying concrete similarity measures. Due to the high variability of projects in general and the limited scope of the project at hand, the focus is furthermore put on software development projects. For this purpose GitHub, which is presented in the next section, will serve as data source.

2.2. Available Data

In order to determine usable similarity measures, the data available regarding software development projects had to be analyzed first. For some time now, online code hosting platforms have experienced a rise in popularity and host a plethora of freely available open source projects. Two especially well-known platforms are GitHub[1] and Sourceforge[2]. As the former hosts more than 52 million projects [11] (compared to more than 430,000 hosted by Sourceforge [12]), it was considered to be a suitable data source.

[1]`https://github.com/`

[2]`https://sourceforge.net/`

2.2.1. GitHub

Founded in 2007, GitHub has established itself as one of the most well-known code hosting platforms for version control and collaboration. According to [11], more than 19 million people are registered and work together on more than 52 million projects. Amongst the latter are popular ventures like the JavaScript-based web application framework AngularJS[3] or Apple's programming language Swift[4].

GitHub provides an extensive application programming interface (API) that allows to retrieve information about virtually any public repository in a programmatic way using representational state transfer (REST) [13]. Access is limited to 1,000 calls per hour for unregistered and 5,000 calls per hour for registered users.

The data available from the API is too extensive to give a complete enumeration, but some of the more interesting informations that can be retrieved are lists of public repositories, of users starring[5] a project or of all users contributing (i.e. actually working on the project) to the project.

Unfortunately, the API has some limitations especially in the context of the call rate limit. For example, upon retrieving the list of all stargazers[6] of a project, one request yields a maximum of 100 entries. Subsequent requests need to include a page number to get access to the next 100 entries. As there are numerous projects with tens of thousands of stars the rate limit can be reached very quickly. For example, retrieving all starrers of the front-end

[3]`https://github.com/angular/angular.js`

[4]`https://github.com/apple/swift`

[5]*Starring* is the process of bookmarking a particular project.

[6]A *stargazer* is the person who starred a particular project.

web framework Bootstrap[7], which has more than 100,000 stars, would take a minimum of 1,000 calls to the API.

2.2.2. GHTorrent

As Gousios [14] points out, the vast amount of available data makes GitHub an attractive research target. However, the rate limit for single users limits the API's capability to serve as data source for research regarding, for example, big data. He therefore started the GHTorrent project, which is collecting data for all public projects available on GitHub by employing numerous API keys donated by registered GitHub users.

In its current version, the GHTorrent project provides the following services [15]:

- Querying MongoDB programmatically
- Querying MySQL through a web interface
- Querying MySQL programmatically
- Streaming of entries in MongoDB and MySQL

As can be deduced by the listed services, GHTorrent maintains an internal storage of the data both in a relational database (MySQL[8]) as well as a document-oriented one (MongoDB[9]). The complete database schema of the MySQL database can be found in the Appendix (Figure 6).

While the GitHub API's rate limit can be circumvented by the usage of the GHTorrent projects, some limitations apply to the collected data that one needs to be aware of for special use cases [16]. Most notable is

[7]`https://github.com/twbs/bootstrap`
[8]`https://www.mysql.com/`
[9]`https://www.mongodb.com/`

the limitation to first order dependencies, which affects, for example, the followers of a project's contributers, for which the next level (i.e. followers following followers) is not stored. Other limitations are related to the way dumps are retrieved and stored, leading to potentially incomplete dumps.

3. Defining Similarity Measures

The previous chapter gave an overview of the theoretical background and the data available for repositories hosted on GitHub. In this chapter, we introduce a set of similarity measures defined based on these findings and the available data. They can be seen as some sort of hypotheses and are applied and statistically tested in Section 4 of this paper. In the subsequent sections each similarity measure is formally defined and verbally described. We want to point out that some of these measures are only partially based on literature and also involve our own assumptions about project similarity, which are based on several years of experience in software development.

3.1. Star-Based Similarity

The already existing project ("gazer"[10] by Andrei Kashcha), which uses the metric shown in Equation (1), provided us with an appropriate starting point for defining our similarity measures.

$$similarity(A, B) = \frac{2 * |stars(A) \bigcap stars(B)|}{|stars(A)| + |stars(B)|} \tag{1}$$

We modified Kashcha's approach to capture the actual percentage of shared stargazers of a project A on the union set of stars of project A and project

[10] `https://github.com/anvaka/gazer/`

B. This produces a value which is, in general, lower than the original measure, as it lacks the multiplier in the numerator. For the denominator, we argue that the amount of stargazers in the union of the two sets is more appropriate, as it does not count one user twice. Hence, our Star-Based Similarity is formally defined in Equation (2).

$$similarity(A, B) = \frac{|stars(A) \bigcap stars(B)|}{|stars(A) \bigcup stars(B)|} \tag{2}$$

This similarity measure is based on the assumption that a user who is interested in a certain kind of software (e.g. code editors) will star several projects dealing with the same topic. However, one should be aware that this formula delivers rather low results when comparing two projects with a substantially different number of stargazers even if the projects are very similar.

3.2. Adjusted-Star-Based Similarity

To remediate the sensitivity of our Star-Based Similarity to different amounts of stargazers, we adapted it in a way that the quotient orients towards the smaller of the two projects. This results in Equation (3).

$$similarity(A, B) = \frac{|stars(A) \bigcap stars(B)|}{min(|stars(A)|, |stars(B)|)} \tag{3}$$

Following this formula, two very similar projects A and B where one is significantly more popular than the other (1000 stars versus 10 stars) that share 10 stargazers would achieve a Adjusted-Star-Based Similarity of 100 % and therefore appropriately reflect a high degree of similarity. In contrast, Star-Based Similarity would achieve a value of only 1 %.

3.3. Star-Time-Based Similarity

Li et al. [17] argue that projects that are starred within a short time frame by the same user tend to be more similar. This is based on the assumption, that a user who is looking for a certain kind of software (e.g. a code editor) will not stop at the first project he encounters, but will rather star it and move on until he found a reasonable number of alternatives that he can compare and choose from, starring all of them along the way. The mathematical definition can be found in Equation (4).

$$similarity(A, B) = \frac{|stars(A) \bigcap_{\{\Delta_t(a,b) \leq 30 | a \in A, b \in B\}} stars(B)|}{|stars(A) \bigcap stars(B)|} \tag{4}$$

Initially, it appeared that Li et al. [17] arbitrarily set the time frame within which two projects had to be starred to be considered to 30 minutes. Nevertheless, we performed some tests with different values which showed us that the first half hour actually has the highest increase of the measure's value, while higher values (from 60 minutes to 48 hours) only showed a marginal increase.

Still, one possible limitation of this similarity measure could be that developers who spend more time on looking for related and similar repositories are not included, leading to a similarity value lower than it actually is.

3.4. Language-Based Similarity

We consider used programming languages within GitHub as another indicator for repository similarity. In our opinion, it is more likely that programmers are able to learn from and reuse code written in the same programming languages. Therefore, we include the measure as formulated in Equation (5) which aims to depict aspects of the language similarity.

$$similarity(A, B) = \frac{|languages(A) \bigcap languages(B)|}{|languages(A) \bigcup languages(B)|} \tag{5}$$

However, we should keep in mind that considering only the language similarity will very likely lead to misleading results because there will be many considerably different projects that use the same programming languages. Hence, this measure should only be used in combination with other similarity measures.

3.5. Contributor-Based Similarity

Dabbish et al. [18] focus on the social aspect of collaborative coding platforms like GitHub. One interviewed developers stated: "When I find a project that solves a problem that I had and I'm going to continue to have then I will watch it". We argue that the same applies for active contributions because developers will focus on a certain domain. As a consequence, they will rather contribute to similar projects. The amount of shared contributors between two projects is calculated by Equation (6).

$$similarity(A, B) = \frac{|contributors(A) \bigcap contributors(B)|}{|contributors(A) \bigcup contributors(B)|} \tag{6}$$

A contributor is thereby defined as a person who either authored a commit to, or reported an issue with a project, or both. Again, we want to point out the necessity to combine this measure with other similarity measures presented in this section.

3.6. Branch-Based Similarity

In order to also include an aspect that takes the behavior of a repository's contributors into account, we propose the Branch-Based Similarity measure shown in Equation (7). It sets the number of two projects' branches into proportion.

$$similarity(A, B) = \frac{min(|branches(A)|, |branches(B)|)}{max(|branches(A)|, |branches(B)|)} \tag{7}$$

Once again, also this measure should be seen as one part of an overall similarity measure.

3.7. Complexity-Based Similarity

We argue that projects tend to get more complex over time if development continues, e.g. by adding new features to keep the software alive and healthy. We argue that a new project on GitHub usually is no adequate reference for an established and more complex one, which is reflected in Equation (8).

$$\begin{aligned} similarity(A, B) = 0.5 * \frac{min(|issues(A)|, |issues(B)|)}{max(|issues(A)|, |issues(B)|)} \\ + 0.5 * \frac{min(|bytes(A)|, |bytes(B)|)}{max(|bytes(A)|, |bytes(B)|)} \end{aligned} \tag{8}$$

Both number of issues (the older or more complex a project is, the more issues arise) and size of the code (the assumption behind that being more code means higher complexity) determine the final outcome of this measure in equal parts. Including this aspect ensures that the similarity degree is reduced according to the difference in complexity.

4. Research Method

In order to determine the actual similarity of two projects, we decided to generate a prediction model. The values of the similarity measures defined in the previous section thereby serve as independent variables for the model. As there is not data readily available defining the actual, objective similarity for any two projects, the dependent variable was determined by us in a subjective way as described in Section 4.1. Section 4.2 then describes how calculation of the similarity scores was performed and the necessary data obtained, before we determine the actual prediction model in Section 4.3.

4.1. Determining the Dependent Variable

To generate a sufficient baseline as input for our prediction model, we decided to draw a random sample of 21 projects from GitHub. This was done using GHTorrent's web-based database access[11], where a random subset of 21 projects was drawn. To provide a good basis for both the similarity measures as well as the subjective rating, selected projects where limited to more than 1000 stars and a creation date not before 2012.

Subsequently, we defined classes of similarity and certain criteria to base our subjective rating on and make it comparable. The classes are split based on two main characteristics, purpose and technology. The purpose of a project is mainly defined by the main aim of the developed software, examples being "text editor", "code editor" or "library". With technology of a project we aimed at the technology stack behind a software. An application based on JavaScript would therefore be identified as "web technology" and considered similar to an application based on HTML, while a "native

[11]`http://ghtorrent.org/dblite/`

technology", for example projects being developed in C, would be considered different. By combining all possibilities we arrived at four different classes listed in Table 1. Two projects are thereby considered to be more similar if they share the same purpose, while the used technology only plays a subordinate role.

Table 1: Description of subjective similarity criteria

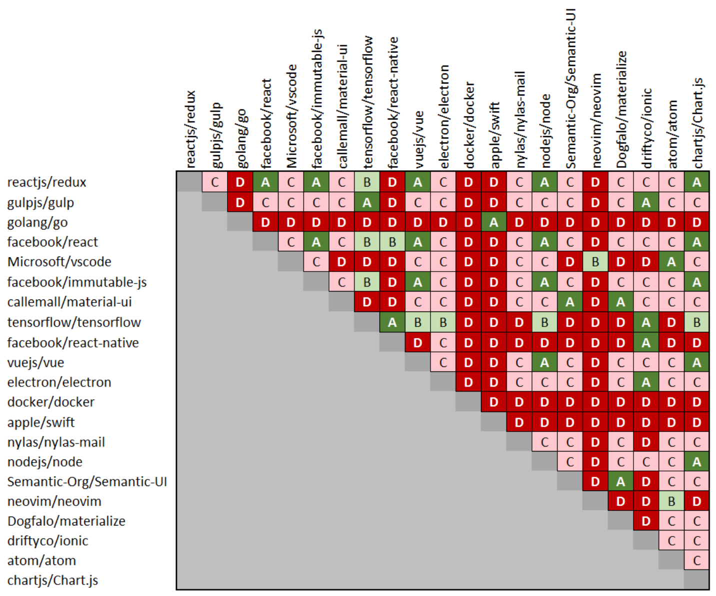
Class	**Name**	**Criteria**
A	similar	same purpose, same technology
B	rather similar	same purpose, different technology
C	rather different	different purpose, same technology
D	different	different purpose, different technology

By applying the listed criteria, 210 pairwise comparisons were performed by all four authors. The individual results were then compared and the final subjective similarity rating was decided upon based on a majority vote (see Figure 1).

4.2. Data Acquisition

In order to gather all relevant data from the sources described in Section 2.2 and calculate the similarity measures defined in Section 3, we developed a tool using the programming language Java, the Project Similarity Calculator (PSC)[12]. The subjective similarities defined earlier serve, together with the names of the compared projects, as input for the tool. Necessary metadata for all 21 projects is fetched from GHTorrent and the GitHub API, before pairwise calculation of the similarity measures is per-

[12]The source code can be downloaded from `https://owncloud.raffel.digital/s/gZ1p3IOy321mDZV` or by contacting one of the authors per email.



	reactjs/redux	gulpjs/gulp	golang/go	facebook/react	Microsoft/vscode	facebook/immutable-js	callemall/material-ui	tensorflow/tensorflow	facebook/react-native	vuejs/vue	electron/electron	docker/docker	apple/swift	nylas/nylas-mail	nodejs/node	Semantic-Org/Semantic-UI	neovim/neovim	Dogfalo/materialize	driftyco/ionic	atom/atom	chartjs/Chart.js
reactjs/redux		C	D	A	C	A	C	B	D	A	C	D	D	C	A	C	D	C	C	C	A
gulpjs/gulp			D	C	C	C	C	A	D	C	C	D	D	C	C	C	D	C	A	C	C
golang/go				D	D	D	D	D	D	D	D	D	A	D	D	D	D	D	D	D	D
facebook/react					C	A	C	B	B	A	C	D	D	C	A	C	D	C	C	C	A
Microsoft/vscode						C	D	D	D	C	C	D	D	C	C	D	B	D	D	A	C
facebook/immutable-js							C	B	D	A	C	D	D	C	A	C	D	C	C	C	A
callemall/material-ui								D	D	C	C	D	D	C	C	A	D	A	C	C	C
tensorflow/tensorflow									A	B	B	D	D	D	B	D	D	D	A	D	B
facebook/react-native										D	C	D	D	D	D	D	D	D	A	D	D
vuejs/vue											C	D	D	C	A	C	D	C	C	C	A
electron/electron												D	D	C	C	C	D	C	A	C	C
docker/docker													D	D	D	D	D	D	D	D	D
apple/swift														D	D	D	D	D	D	D	D
nylas/nylas-mail															C	C	D	C	D	C	C
nodejs/node																C	D	C	C	C	A
Semantic-Org/Semantic-UI																	D	A	D	C	C
neovim/neovim																		D	D	B	D
Dogfalo/materialize																			D	C	C
driftyco/ionic																				C	C
atom/atom																					C
chartjs/Chart.js																					

Figure 1: Subjective similarity rating used as dependent variable

formed. The calculated stores are then written into a comma separated value (CSV) file. Each line contains all scores of one pairwise comparison as well as the initial subjective similarity rating. The overall structure of the PSC is shown in Figure 2 and the subsequent sections explain the behavior of each component.

In the project, the `at.ac.wu.is.psc.ProjectSimilarityCalculator` class serves as starting point for the tool and links all components. We suggest to start exploration of PSC's code there.

4.2.1. Acquiring GHTorrent Data

In order to acquire the necessary data for calculation of the similarity scores, GHTorrent provided the most convenient alternative to circumvent the rate limitation of GitHub's API (see Section 2.2.1). To be able to directly connect to the GHTorrent server and gain access to the MySQL database, a

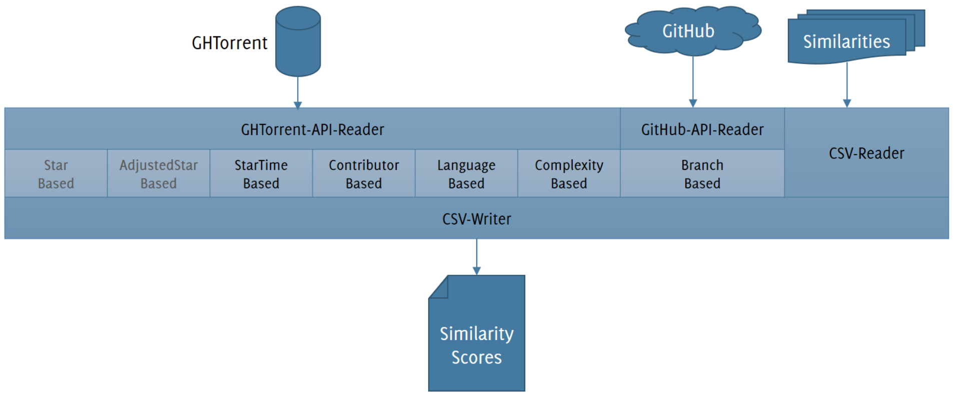


Figure 2: Components of the Project Similarity Calculator

SSH key pair has to be generated and the public key needs to be given to the GHTorrent team by pushing it onto their GitHub repository[13]. Following this process enabled us to establish a tunnel to the GHTorrent server, thus being able to access the MySQL database directly by using a specific port.

Out of the full database schema of the GHTorrent database (which can be seen in the Appendix, Figure 6), only the tables and fields shown in Figure 3 are actually used to load the metadata for the specified projects.

In the project, the relevant classes managing database access can be found in the package `at.ac.wu.is.psc.database` while classes modeling the actual entities are stored in the package `at.ac.wu.is.psc.model`.

4.2.2. Accessing GitHub's API

As the data that is necessary to compute the Branch-Based Similarity (see Section 3.6) is not readily available within GHTorrent (it would require extensive matching of commits to rebuild the branch structure), the GitHub API was used to obtain the number of branches for each project. Although

[13]`https://github.com/ghtorrent/ghtorrent.org`

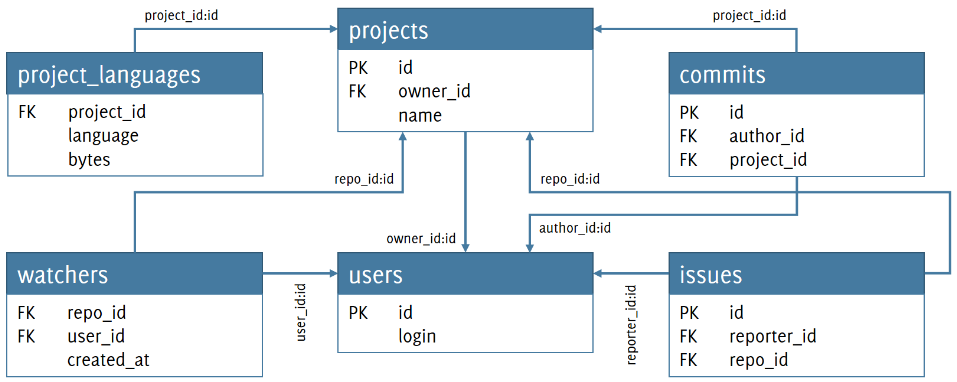


Figure 3: Database schema of GHTorrent (used fields and relations)

most projects have less than 100 branches and the whole list can therefore be retrieved with only one API call per project, obtaining a personal access token is still advisable to raise the limit of available calls per hour[14].

In the project, the class `at.ac.wu.is.psc.api.GithubAPI` is responsible for fetching the relevant branch data from the API available via `https://api.github.com/repos/user_name/project_name/branches`. Once acquired, the personal access token needs to be put inside the code of this class.

4.2.3. Calculating Similarity Measures

Each similarity measure is calculated based on the metadata retrieved and stored into the respective model classes beforehand. By making use of Java's integrated `java.util.Set`-classes, set-based operations necessary for virtually all defined similarity measures (i.e. union and intersection) are readily available.

[14]This can be done by registering a account with GitHub and generating a Personal Access Tokens on `https://github.com/settings/tokens`

In the project, each similarity measure is defined in its own class located in the package `at.ac.wu.is.psc.method`. This ensures a well-defined structure as well as easy adoption or extension of similarity measures.

4.2.4. Exporting Data for Statistical Analysis

After calculation of all necessary similarity measures is completed, the results are written into a CSV file for further processing. An exemplary export of the PSC can be seen in the Appendix (Listing 1). Table 2 lists the calculated measures for some projects in order to generate a better understanding for their possible relationship to the actual (subjective) similarity.

Table 2: Actual similarity measures for several projects

Similarity Measure	reactjs/redux gulpjs/gulp	reactjs/redux golang/go	reactjs/redux facebook/react	reactjs/redux vuejs/vue
Rating	C	D	A	C
Star-Based	7.07 %	4.00 %	11.24 %	7.58 %
Adjusted-Star-Based	15.83 %	10.62 %	45.93 %	23.85 %
Star-Time-Based	4.45 %	2.89 %	19.58 %	7.93 %
Contributor-Based	1.20 %	0.18 %	3.08 %	0.27 %
Language-Based	0.00	0.00 %	0.00 %	0.00 %
Complexity-Based	44.44 %	40.91 %	29.03 %	52.94 %
Branch-Based	27.45 %	4.59 %	5.55 %	15.74 %

4.3. Statistical Analysis

Determining the similarity of two software projects can be seen as classification problem. Therefore, we created a classification model using the analysis data produced above. This prediction model takes continuous values as input and produces a discrete outcome (i.e. the class assigned to two projects with specific similarity measures). In order to find the most suitable classifier for our purpose we used several well-known statistical techniques. The used methods for fitting a classification model are:

- Multinomial Logistic Regression (Linear features)
- Multinomial Logistic Regression (Polynomial features)
- Linear Discriminant Analysis
- Naïve Bayes
- Nearest Neighbor Classifier

After evaluating the performance of each classifier and comparing their results, we chose the one which tends to be the most promising for classifying two projects into our predefined classes (see Section 4.1).

4.4. Results

This section describes the approach of conducting the statistical analysis and the progress made by using different techniques. Starting point is the analysis data produced in Section 4.2.4 above. This data acts as input for the classification model. The data set comprises 210 entries. Each entry represents a comparison of two projects. The comparison of two software projects comprises the measures between them, conducted by our defined similarity metrics, and a class, which is a subjective classification executed by ourselves (see Section 4.1).

To sum up, the similarity measures are used as predictors to generate the class feature as response (dependent) variable. In other words, we use the similarity measures as independent features and classify the comparison of two software projects based on the value of each predictor.

4.4.1. Random Train/Test set

After splitting up the features in a response and predictor space the data is shuffled 50 times and divided into a train and test set. This approach helps to compare the performance of all available classifiers based on accuracy, precision, recall and the combination of the latter two (F1-score). Each classifier is trained on 80 % of the whole data and tested on the remaining 20 %. The results are depicted in Table 3.

Table 3: Results of classification metrics using train/test data

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
MLR (Linear features)	0.62	0.54	0.62	0.57
MLR (Polynomial features)	0.71	0.71	0.71	0.71
Linear Discriminant Analysis	0.62	0.54	0.62	0.57
Naïve Bayes	0.62	0.48	0.52	0.49
Nearest Neighbor Classifier	0.40	0.53	0.40	0.40

Table 3 summarizes all results regarding the used classification methods. It can be said that the logistic regression model delivers the best predictions. We use logistic regression for a multiclass problem, therefore, we say it is a multinomial logistic regression. To extend this model and to get a holistic view in terms of interacting features, we also trained this model on polynomial features. The idea behind this approach is to think about predictors not only individually influencing the response variable, but also combined with another predictor. Hence, we transformed the linear features into polynomial features with a degree of three. This results in a set of 55 variables, which are then used as predictors for fitting a logistic regression model. Table 3 shows an improvement of the metrics mentioned before if we use logistic regression with the polynomial features to the degree of three.

4.4.2. 5-Fold Cross Validation

The second approach used to compare the results of the classification methods is 5-fold cross validation. This technique splits the data into five so-called "folds". Each fold is used as test set once. The remaining folds are used to train the respective classifier. After five train/test executions, the results are summed up and divided by the number of folds. This delivers the mean value of each metric tested on each part of the data once. The respective results are displayed in table 4.

Table 4: Results of classification metrics using cross validation

Classifier	**Accuracy**	**Precision**	**Recall**	**F1-score**
MLR (Linear features)	0.53	0.46	0.53	0.47
MLR (Polynomial features)	0.55	0.56	0.55	0.55
Linear Discriminant Analysis	0.54	0.48	0.54	0.48
Naïve Bayes	0.44	0.45	0.44	0.40
Nearest Neighbor Classifier	0.28	0.33	0.28	0.27

4.4.3. Receiver Operating Characteristic

To get an understanding about which of the classification methods are most suitable for the problem of determining similar projects, we need to dive deeper into the analysis data and their results. This can be done by plotting the receiver operating characteristic (ROC) curve of each classifier and computing the area under the curve for each class. In order to do such an analysis, the data was cross validated and the ROC curves were created for each fold of a specific class. Subsequently, the area under the curve (AUC) scores are summed up and divided by five resulting in a mean ROC curve for a specific class. In addition, the mean ROC curves are generated for the remaining classes.

This results in a diagram created to analyze the performance of the classifier (see Figure 4).

The last step is to add the micro and macro ROC curves of the classifier in order to create a comprehensive understanding of the performance and to decide whether a classifier is better or not. The values of the AUC are summarized in Table 5 below.

Table 5: Cross validated mean AUC for each class

Classifier	**Class A**	**Class B**	**Class C**	**Class D**
MLR (Linear features)	0.66	0.58	0.80	0.80
MLR (Polynomial features)	0.59	0.69	0.77	0.80
Linear Discriminant Analysis	0.65	0.63	0.77	0.82
Naïve Bayes	0.66	0.57	0.78	0.79
Nearest Neighbor Classifier	0.48	0.65	0.62	0.63

4.5. Discussion

The aim of this part of the paper is to come up with the most interesting findings of the conducted statistical analysis. For discussing these results, one should keep the main aim of this research in mind. The overarching goal formulated at the beginning of this work is to find similar software projects by applying the set of similarity measures (defined in Section 3) and, as a result, improve the quality of ongoing projects by learning from other similar projects. Therefore, we conducted a classification task which should support the predictions of the similarity between two projects.

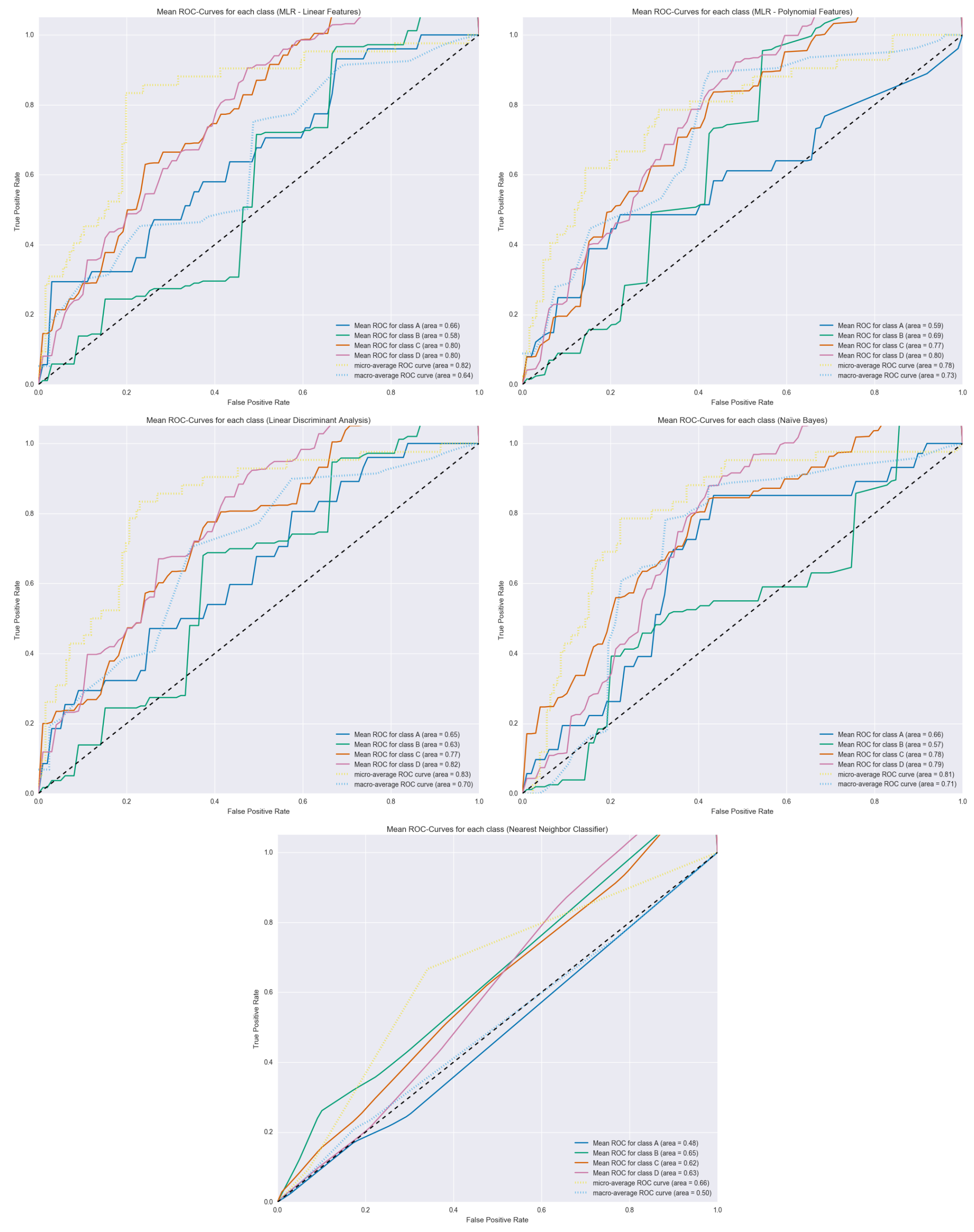


Figure 4: Cross validated ROC curves for each class plotted for every classifier

4.5.1. The Approach in a Nutshell

In a nutshell, the approach of solving the problem of finding similar and different software repositories on GitHub was a follows. First, each project was rated against each other project subjectively. We decided to use a more fine-granular level of classification in order to avoid misclassification by our own error prone thinking. After that, the analysis data was created by applying the similarity measures on different project comparisons. At the end, prediction models were trained using several statistical methods for classification and the produced analysis data described in section 4.2.

4.5.2. Comparing Basic Classification Metrics

Now the different classifiers need to be investigated in order to determine if they make appropriate predictions or if it is even possible to predict a certain level of similarity based on the set of measures that were used to generate the data set. This can be done by investigating the results of Section 4.4. To get a rough picture of which model delivers valuable predictions we looked at the cross-validated accuracy, precision, recall and F1-score of each classifier. Table 4 shows that the multinomial logistic regression (MLR) with polynomial features is the most accurate and precise model. Also the F1-score is the best one compared to the others. Further findings are that the MLR with linear features performs the same as linear discriminant analysis, whereas naïve bayes and nearest neighbor classifier can be considered as inappropriate for our purpose. Nevertheless, the results of MLR using polynomial features are not as good as expected. The values of the metrics are above the 50 % threshold, but the results look more or less like a random classification of the projects. Therefore, we need to do further investigations on the classification process and the classes we used for that problem.

4.5.3. Comparing ROC curves and AUC of Classes

The ROC curves of a classifier deliver a pretty good picture of how classes are predicted. Interpreting these curves uncovers the problems with predicting classes for project comparisons. Figure 4 shows each of the classifier and the corresponding ROC curves for each class. The values are cross-validated and the mean ROC has been plotted for the encoded classes A, B, C and D. As can be seen in the plots, the classes C and D are predicted pretty well. Projects which are different according to our assessment are also considered as different by the classifiers. This is also reflected by the AUC scores summarized in table 5, which are all (except the nearest neighbor classifier) above the 75 % threshold. These are quite good values for predicting a class. Nevertheless, there is a problem with predicting similar projects. The ROC curves of class A and B are near the diagonal, which is an indicator for a random classification. Also the AUC scores from 48 % (the weakest) to 69 % (the strongest) are not the best. Overall, it can be said that if two projects are similar or rather similar, the classifiers are misclassifying them in most cases. This is a problem, because of the overall goal of determining if two projects are similar or different. One solution of that problem is the use of polynomial features, which can be seen in the second plot of Figure 4. The ROC curve of class B and the corresponding AUC of 69 % improved in comparison to the other classification models. This helps to improve the distinction between similar and different software projects. Nevertheless, there is still a problem with predicting very similar projects. In many cases, the used statistical methods classify similar projects as different, which is a big issue keeping in mind that the essence of this research is to find similar software project repositories.

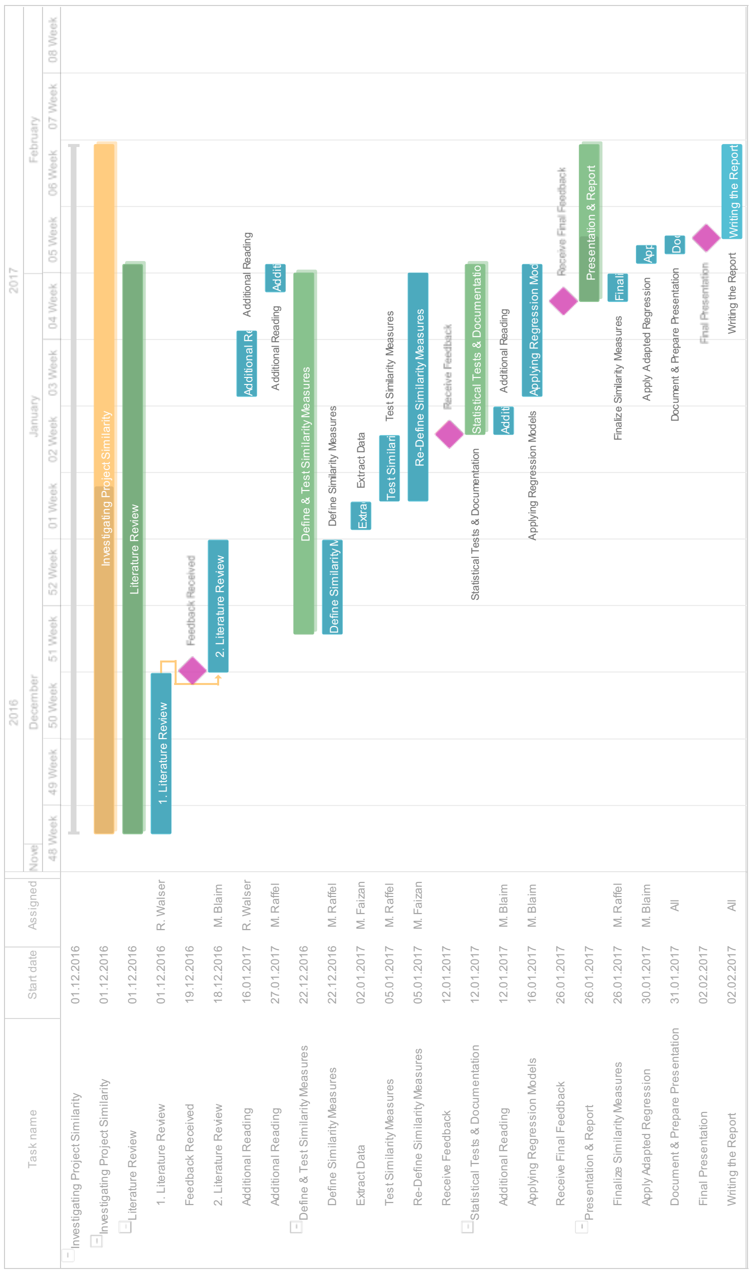
5. Conclusion

In this paper, we investigated a set of measures for assessing project similarity of projects available on GitHub. To do so, we defined a set of similarity measures and developed a Java-based project similarity calculator which automatically extracts relevant data from GitHub and GHTorrent. Furthermore, results of our statistical analysis indicate that the prediction model reliably classifies different or rather different software projects.

However, for similar or rather similar projects, the defined prediction model offers room for improvement. This could be caused by the occurrence of different aspects of similarity. A more precise and accurate classification of similarity might only be possible for different and more specific types of similarity (e.g. technical similarity, content-related purpose similarity). However, some initial tests in this direction did not bring the expected success. Due to the limited scope of this paper, the prediction model could only be trained and tested with a limited number of GitHub repositories. Furthermore, the personal and subjective similarity classification might be faulty and therefore bias the results.

Possible further work includes not only an extension of the data set, but also splitting up the measure ”similarity” in an appropriate and justifiable way. A concrete set of criteria should be elaborated on in order to illuminate the different aspects of project similarity and increase reliability of the respective similarity predictions. Additionally, the number of considered projects could be increased as well as the number of similarity assessments per project.

Appendix



Task name	Start date	Assigned
Investigating Project Similarity	01.12.2016	
Investigating Project Similarity	01.12.2016	
Literature Review	01.12.2016	
1. Literature Review	01.12.2016	R. Walser
Feedback Received	19.12.2016	
2. Literature Review	18.12.2016	M. Blaim
Additional Reading	16.01.2017	R. Walser
Additional Reading	27.01.2017	M. Raffel
Define & Test Similarity Measures	22.12.2016	
Define Similarity Measures	22.12.2016	M. Raffel
Extract Data	02.01.2017	M. Faizan
Test Similarity Measures	05.01.2017	M. Raffel
Re-Define Similarity Measures	05.01.2017	M. Faizan
Receive Feedback	12.01.2017	
Statistical Tests & Documentation	12.01.2017	
Additional Reading	12.01.2017	M. Blaim
Applying Regression Models	16.01.2017	M. Blaim
Receive Final Feedback	26.01.2017	
Presentation & Report	26.01.2017	
Finalize Similarity Measures	26.01.2017	M. Raffel
Apply Adapted Regression	30.01.2017	M. Blaim
Document & Prepare Presentation	31.01.2017	All
Final Presentation	02.02.2017	
Writing the Report	02.02.2017	All

Figure 5: GANTT-Chart showing work plan and division of labor

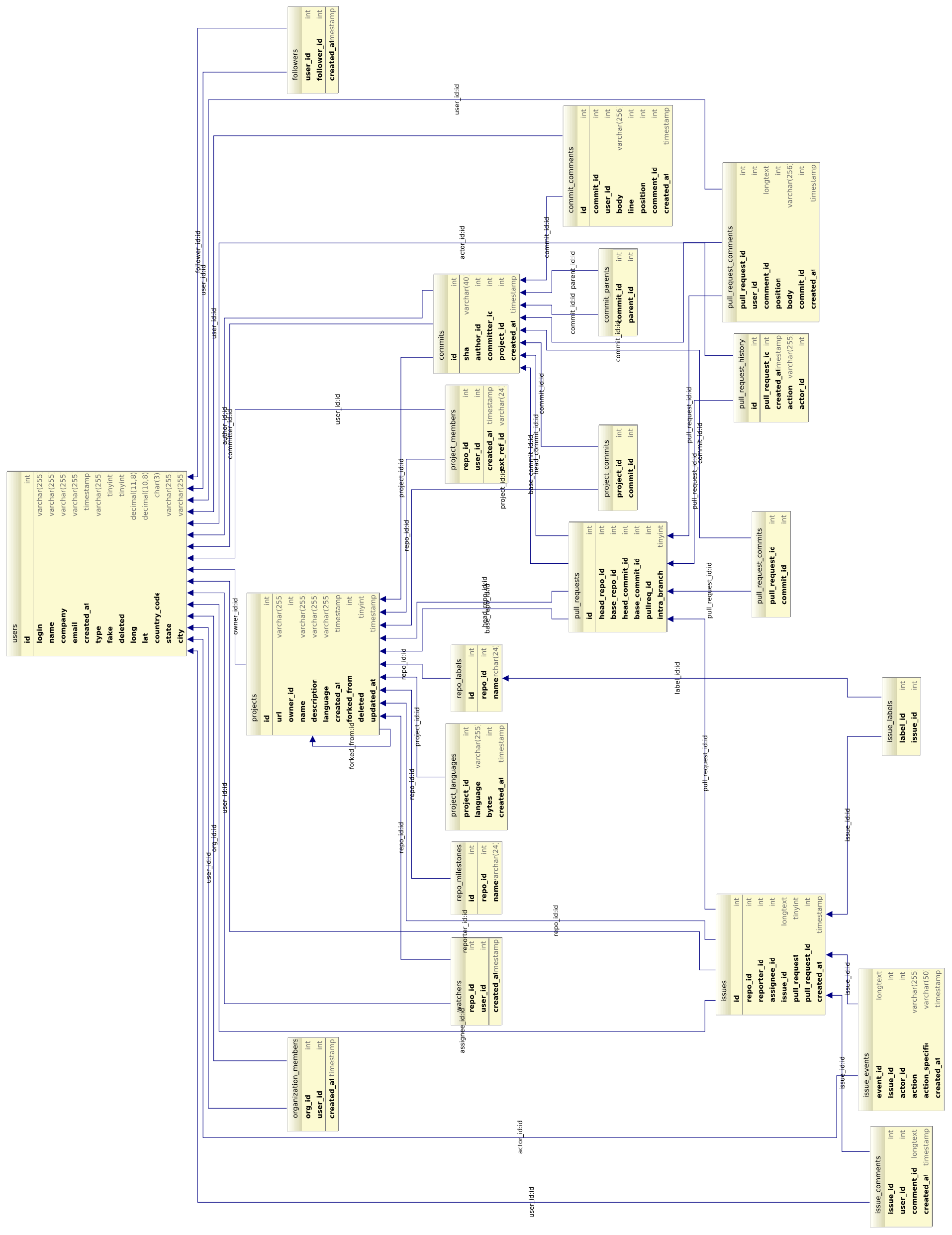


Figure 6: Database schema of GHTorrent (complete)

Listing 1: Content of the exported CSV file (rounded to two decimal places)

```
A,B,Star,AdjustedStar,StarTime,Contributor,Language,Branch,Complexity,Rating
reactjs/redux,gulpjs/gulp,7.06,15.83,4.45,1.20,0.00,44.44,27.45,C
reactjs/redux,golang/go,4.00,10.62,2.89,0.18,0.00,40.91,4.59,D
reactjs/redux,facebook/react,11.24,45.93,19.58,3.08,0.00,29.03,5.55,A
reactjs/redux,Microsoft/vscode,4.87,11.74,3.33,0.42,0.00,10.59,2.51,C
reactjs/redux,facebook/immutable-js,9.55,18.95,8.03,3.24,0.00,33.33,46.56,A
reactjs/redux,callemall/material-ui,8.95,21.06,5.07,1.95,0.00,66.67,8.28,C
reactjs/redux,tensorflow/tensorflow,3.66,14.41,2.86,0.06,0.00,47.37,6.85,B
reactjs/redux,facebook/react-native,8.30,31.20,8.24,1.13,0.00,17.65,4.06,D
reactjs/redux,vuejs/vue,7.58,23.85,7.93,0.27,0.00,52.94,15.74,A
reactjs/redux,electron/electron,6.30,12.38,6.26,0.40,0.00,47.37,12.71,C
reactjs/redux,docker/docker,4.18,15.33,3.10,0.23,0.00,55.56,1.95,D
...
```

References

[1] P. Drineas, I. Kerenidis, P. Raghavan, Competitive recommendation systems, in: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM, pp. 82–90.

[2] G. Adomavicius, A. Tuzhilin, Context-aware recommender systems, in: Recommender systems handbook, Springer, 2015, pp. 191–226.

[3] M. Robillard, R. Walker, T. Zimmermann, Recommendation systems for software engineering, IEEE software 27 (2010) 80–86.

[4] T. Cooke-Davies, The "real" success factors on projects, International journal of project management 20 (2002) 185–190.

[5] W. Belassi, O. I. Tukel, A new framework for determining critical success/failure factors in projects, International journal of project management 14 (1996) 141–151.

[6] J. Verner, J. Sampson, N. Cerpa, What factors lead to software project failure?, in: Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on, IEEE, pp. 71–80.

[7] M. Shepperd, C. Schofield, Estimating software project effort using analogies, IEEE Transactions on software engineering 23 (1997) 736–743.

[8] Y.-F. Li, M. Xie, T. N. Goh, A study of project selection and feature weighting for analogy based software cost estimation, Journal of Systems and Software 82 (2009) 241–252.

[9] M. Azzeh, D. Neagu, P. Cowling, Software project similarity measurement based on fuzzy c-means, in: International Conference on software process, Springer, pp. 123–134.

[10] R. Schmidt, K. Lyytinen, P. C. Mark Keil, Identifying software project risks: An international delphi study, Journal of management information systems 17 (2001) 5–36.

[11] GitHub - About, 2017. `https://github.com/about` (visited on 02/12/2017).

[12] Sourceforge - About, 2017. `https://sourceforge.net/about` (visited on 02/12/2017).

[13] GitHub API v3 — GitHub Developer Network, `https://developer.github.com/v3/` (visited on 02/12/2017).

[14] G. Gousios, The GHTorrent Dataset and Tool Suite, in: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, IEEE Press, Piscataway, NJ, USA, 2013, pp. 233–236.

[15] G. Gousios, GHTorrent services, `http://ghtorrent.org/services.html` (visited on 02/12/2017).

[16] G. Gousios, B. Vasilescu, A. Serebrenik, A. Zaidman, Lean GHTorrent: GitHub Data on Demand, in: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR '14, ACM, pp. 384–387.

[17] Q. Li, Y. Li, P. S. Kochhar, X. Xia, D. Lo, Detecting similar repositories on github (2012).

[18] L. Dabbish, C. Stuart, J. Tsay, J. Herbsleb, Social coding in github: transparency and collaboration in an open software repository, in: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work, ACM, pp. 1277–1286.