Time is Money

A similar repository recommender system that saves development time using tag hierarchies

Andrei-Cristian Ionescu
Supervisors: Dr. Maliheh Izadi, Prof. Dr. Arie van Deursen
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Developers do not want to reinvent the wheel when developing software systems. Open-source software repositories are packed with resources that may assist developers with their work. Since Github enabled repository tagging, a new opportunity arose to help developers find the needed resources tailored to their needs. The current work proposes two similar repository approaches enhanced by a tag hierarchy generation approach that is used in our recommender systems. The first approach provides advanced repository indexing, and it is constructed as a wrapper around the Google Programmable Search, and the second approach is based on the GitHub Search API. During the study, we developed and deployed a fully-fledged framework that allows us to create, label, weigh and evaluate any tag hierarchy and our recommending approaches.

Analyzing the results collected by our evaluation framework, we concluded that the Google Search approach is preferred over the GitHub approach from both accuracy and user perspective point of view. The Google Search approach outperformed the baseline by 18.75%, but also the GitHub Search by more than 100% concerning the MAP (Mean Average Precision) metric.

1 Introduction

Developers do not want to reinvent the wheel when establishing a new software system. Open-source software repositories are packed with resources that may assist developers with their work. Our research intends to assist developers in locating such resources, allowing them to concentrate on the essential functions of an application or system rather than implementing fundamental functionalities from scratch.

Each repository tag englobes knowledge about the application that is developed in a particular repository, and a set of tags represents the architecture, and the overall system characteristics and dependencies [1]. Since GitHub enabled the repository tagging, a new opportunity arose to help the developers to find the needed resources based on their needs.

The currently available literature and tools do not offer many solutions directly related to recommending similar repositories which are tailored and based on the researcher or developer's needs. Recommender systems and tools can identify comparable repositories [2, 3], but they focus on static approaches that can be used just for a small number of repositories (e.g. Java Repositories). Furthermore, some of these approaches are limited to their training data and cannot adapt to newly published repositories or technologies.

During our research, we aim to address the recommendation generality gap in recommending similar repositories. Furthermore, we aim to study the use of tag hierarchies in such a holistic recommender system.

To address various elements of both our hierarchy component and recommender systems, we hope to answer the following research question and sub-questions:

How accurate and useful is a similar repository recommender built on the tags and the hierarchy?

- How useful and accurate is a similar recommender repository based on tags?
- How to identify and create a reliable hierarchy based on tags and the relationships between them? (e.g. Java $\xrightarrow{Framework}$ Spring $\xrightarrow{Testing}$ JUnit)
- How to assess the performance when a hierarchy is used in a similar repository recommender system based on repository tags?
- What search techniques are preferable when it comes to Similar Repository recommender?

To address the tag hierarchy that will be used in the recommender system, we look into the k-graph created by Izadi et al. [4]. The k-graph incorporates the relationships between different technologies. The k-graph is mined, and a dependency hierarchy is created and analyzed using clustering techniques and a self-made app, "Hierarchy Visualiser"[1]. For a similar recommendation system, we used two different approaches: one explores the GitHub Search API (4.3.2), and the other one is based on the Google Programmable Search engine (4.3.3). Both approaches output a ranked list containing repository recommendations based on a search query. The tag hierarchy is integrated into our approaches as a sub-recommending system that recommends topic tags based on a search query.

[1] `https://hierarchy-visualiser.herokuapp.com/`

The similar repository recommender approaches are evaluated using an expert scenario-based evaluation [5]. During the study, an evaluation framework was developed and deployed[2]. The evaluation framework implements the search approaches and the baseline, collects user feedback and retrieves, validates and automatically computes the results.

We demonstrate that the Google approach outperforms GitHub's state-of-the-art "best match" search by 18.75% in terms of MAP (Mean Average Precision). We also discovered that GitHub's Search API approach does not provide a reliable method of recommending similar repositories based on tags and a tag hierarchy, outperforming the Google Approach by more than 100%.

To ensure the replicability of our study, we provide two replication packages. The first replication package contains the front-end[3] of our approaches (e.g. hierarchy data visualisation, labeling, evaluation), and the second one contains the back-end[4] implementation of our approaches (e.g. hierarchy construction, google recommendations, automatic statistics and evaluation).

2 Related Work

A recommender system aims to generate meaningful recommendations for different entities that may interest a group of users. In our case, the users who may benefit from such a recommender system are the developers or researchers searching for programming "tools" to maximize their productivity or make their life easier with existing repositories that may achieve their goals. Similar work was conducted by Izadi et al. [4] that studied the recommendation topic in the context of software repositories using multi-label classification, assigning topics tags for input repositories. They brought a reliable solution for automatically classifying software entities such as GitHub repositories. Their system achieved outstanding performance concerning the topic recommendations with a maximum LRAP score of 0.805 for the LR, TF-IDF model trained in just 30 minutes with a prediction processing time of 0.4 seconds and topics with a high R@5 score of 0.890 can be suggested by their algorithm. Furthermore, the authors concluded that GitHub enabled repository owners to specify the primary aspects of their repositories using a few basic textual phrases. The authors developed several multi-label classifiers to select subjects (topics/tags) for repositories based on their name, description, README files, wiki pages, and file names.

Ponzanelli et al. [2] also studied the same type of recommender system but from a holistic approach. Their recommender system, Libra, tracks web search results, pages visited, and code typed and updated by the developer in the web browser and the IDE. The analysis performed by Libra does not treat the contents of resources as plain text (as Izadi et al. [4]) but instead considers their diverse composition. Their approach aids developers in picking relevant results from a web search by considering the popularity of a particular resource and the developing process as a whole. The authors concluded that Libra supports developers in picking relevant results from a web search by assessing the prominence of a given resource and the complementarity of a result with the obtained knowledge context by holistically examining the contents. Two studies assessed the use of the tool during development and its applications in an industrial setting to determine its utility throughout development.

Izadi et al. [6] further studied the topic of recommender systems using repository tags. The authors focus on two recommender models for labelling software projects considering the semantic link between subjects. To do this, they created a knowledge graph that captures the semantic relationships between repository tags. Their approach outperforms the baselines that neglect these relationships, improving by at least 25% in terms of Average Success Rate. We consider that the information from their knowledge graph can further be "mined" to obtain more data about the relationships between the topics (such as a tag/topic hierarchy).

The authors/developers of RepoPal [3] provide an unique technique for detecting comparable projects on GitHub. Their method is based on three heuristics and two data sources (i.e., GitHub stars and README files) that have not been explored in earlier publications. Using a thousand Java repositories, they build a recommender system named RepoPal that is compared to a previous state-of-the-art solution called CLAN (Closely reLated ApplicatioNs).

RepoPal uses two data sources (i.e., GitHub stars and README files) that might intuitively aid in

[2] https://hierarchy-visualiser.herokuapp.com/#/evaluation

[3] https://github.com/andrei5090/SimilarRepoRecommender/tree/main/webapp

[4] https://github.com/andrei5090/SimilarRepoBackend

identifying related repositories but have not been examined in earlier research. One technique is based only on similarities in API method invocations, while another is based on tags that do not exist in GitHub. To assess RepoPal's efficacy, they conducted experiments on one thousand Java projects on GitHub and compared them to the state-of-the-art technique CLAN. The authors observed that RepoPal outperformed CLAN by 30% in terms of F1-score.

The authors observed that when the star rating of a repository decreases, the retrieval quality of RepoPal may suffer (e.g. recommendation quality decreases). GitHub has many repositories written in programming languages other than Java (e.g., Python, PHP, C++), as well as combinations of various programming languages, but the evaluation focuses only on Java since CLAN was built around Java.

Recommender systems such as the one presented by Ponzanelli et al. [2] offer intelligent assistance to programmers throughout their work. In general, this recommender system focuses primarily on IDEs but not on the general context of a user searching for solutions such as searching on web browsers. The authors developed a recommender system called Libra. Libra is a comprehensive recommender system that augments the web browser with a specialized interactive navigation chart.

Libra tracks web search results, pages visited, and code typed and updated by the developer in both the web browser and the IDE. The analysis performed by Libra does not treat the contents of resources as plain text but instead takes into consideration their diverse composition. It aids developers in picking relevant results from a web search by considering the popularity of a particular resource and the developing process as a whole.

The study offers a good overview of what data is needed to establish a software hierarchy using data directly collected from the users. The paper also provides a framework for analyzing the recommender system from a user perspective (not just a numerical one) using controlled experiments. On the other hand, the controlled experiments use a small number of people (16), and it is not clear what type of users the authors are using in their controlled experiments (they offer an example of university students), but testing the usefulness just on university students may not be the best alternative.

In the work by Lopez et al. [7], the authors propose the concept of utilizing software metrics to assist developers in selecting the libraries that are most suited to their needs. They suggest constructing library comparisons based on metrics taken from various sources, including software repositories, problem tracking systems, and Q&A websites. The authors aim to empower developers to make educated decisions by combining this information and summarizing it on a result page.

The authors found eight metrics that may help assess a particular library's usefulness (popularity, release frequency, issue response and closing times, recency, backwards compatibility, migrations, fault-proneness, performance and security). The authors found that a dedicated web page that summarizes each library based on the metrics mentioned above may improve the programmers' choices when searching what libraries they use in their codebase.

The authors offer selection criteria based on the user perspective (e.g. library usefulness) but also metrics that may be useful in establishing a tag-relationship hierarchy of software repositories (e.g. Java $\xrightarrow{Framework}$ Spring $\xrightarrow{Testing}$ JUnit). One such metric is the "popularity" metric that may help us categorize hierarchies that may have similar trees (e.g. from the popularity metric, the library IziToast is more probable to be used with Vue or React than Nuxt, even if Nuxt is an enhanced version of Vue). The short-come of the paper is the fact that their results are still preliminary and not all their metrics were thoroughly tested yet.

The existing available literature and technologies do not provide many straightforward answers for proposing comparable repositories matched to the needs of the researcher or developer. There are recommender systems and tools that can find comparable repositories [2, 3], but they are static and can only be utilized for a limited number of repositories (e.g. Java Repositories). Our research aims to provide a holistic approach to recommending similar repositories that can automatically recommend new technologies or repositories without any human intervention.

3 Problem Description

The Open Source Software community on GitHub hosts a set of repositories R = $\{r_1, r_2, r_3, r_4, ..., r_n\}$ where r_i represents a software repository. Some software repositories r_i can be characterised by software topics

(tags) added by the repository's developers where $r_i = t_1, t_2, ..., t_n$ where each t_i is a topic tag. Figure 1 and Figure 2 represent similar software repositories with similar, but not identical defined tags.

We aim is to recommend similar software repositories, based on tags and hierarchy. The hierarchy is composed by a recursive data structure (tree)
$H = \{\{n_1, w_1, [t_1 1, t_1 2, ..., t_1 n], c_1 : [n_2, w_2, [t_2 1, t_2 2, ..., t_2 n], c_1 : \{...\}]..\}$ where n_1 represents the name of the node in the hierarchy, w_1 represents the weight of the node and $[t_1 1, t_1 2, ..., t_1 n]$ a list of topics available in the node and c_1 that represents a list of the children of a node.

4 Repository recommendation

In this section, we first analyze the Hierarchy building topic, and then we look into how such a hierarchy can be visualized and labelled with the purpose of repository recommendation. Finally, we use the hierarchy combined with search engines such as GitHub Search API or Google Search to look into how such a Hierarchy can be used to recommend similar repositories. Figure 3 illustrates the overall workflow of our repository recommendation.

4.1 The Hierarchy

In order to achieve our goal of recommending similar repositories, we established that a tag (topic) hierarchy is needed to organize the topics featured and curated on GitHub. Two approaches were used in order to create such a hierarchy. The first try is using expert knowledge and manual work to characterize and establish relationships between the tags that were further used to create a hierarchy. The second approach is using an automatic approach to characterise the topic/tags. Each topic representation is then used with various clustering techniques to establish a topic hierarchy.

4.1.1 Manual built Hierarchy

The manually built hierarchy considered expert knowledge about a selection of GitHub Topics established by Izadi et al. [6]. Each group of topics was manually labelled into sub-groups and then arranged into a dependency hierarchy (e.g. Frontend $\xrightarrow{Framework}$ (React, JavaFX) $\xrightarrow{Language}$ (Javascript, Java)). We concluded that the manual creation of such Hierarchies is unreliable since the manual work is tedious, prone to error and hard to update when new technologies are considered in a pre-existing Hierarchy.

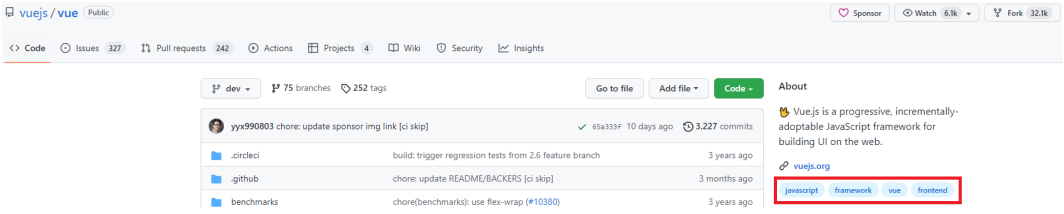

Figure 1: Vue Software Repository with highlighted tags

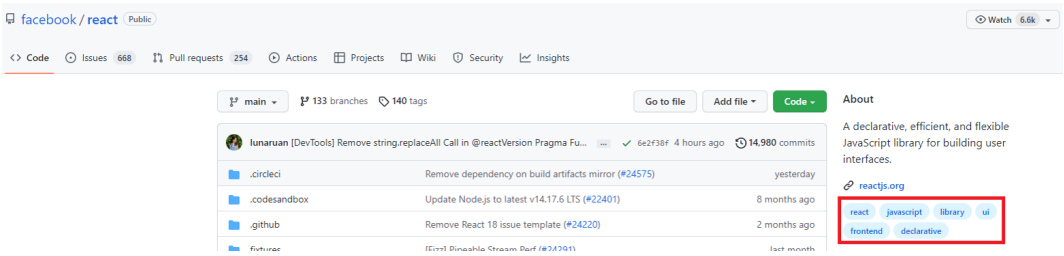

Figure 2: React Software Repository with highlighted tags

4.1.2 Automated built Hierarchy

The automatic build Hierarchy considers the SED-KGraph constructed by Izadi et al. [6]. Each topic (tag) is represented in the graph as a node, where each inbound or outbound edge represents the relationship between the tags. One such an example is:

- AWS $\xrightarrow{is-a}$ service
- AWS $\xrightarrow{is-used-in-field}$ cloud-computing
- AWS $\xrightarrow{provided-by}$ amazon
- AWS $\xrightarrow{provides-functionality}$ as-a-service
- AWS $\xrightarrow{provides-functionality}$ distribution
- AWS $\xrightarrow{provides-functionality}$ hosting
- Terraform $\xrightarrow{works-with}$ AWS
- Amazon $\xrightarrow{provides-product}$ AWS

Each tag is represented as a matrix M $=$

$$\begin{bmatrix} & \underline{AWS} & \underline{ADA} & \underline{AJAX} & \dots & \underline{ZERONET} \\ \xrightarrow{is-a} & x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ \xrightarrow{is-based-on} & x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \xrightarrow{works-with} & x_{m1} & x_{m2} & x_{m3} & \dots & x_{mn} \end{bmatrix}$$

where each row is represented by a relationship in the k-graph related to the current tag (e.g. $\xrightarrow{is-a}$), and

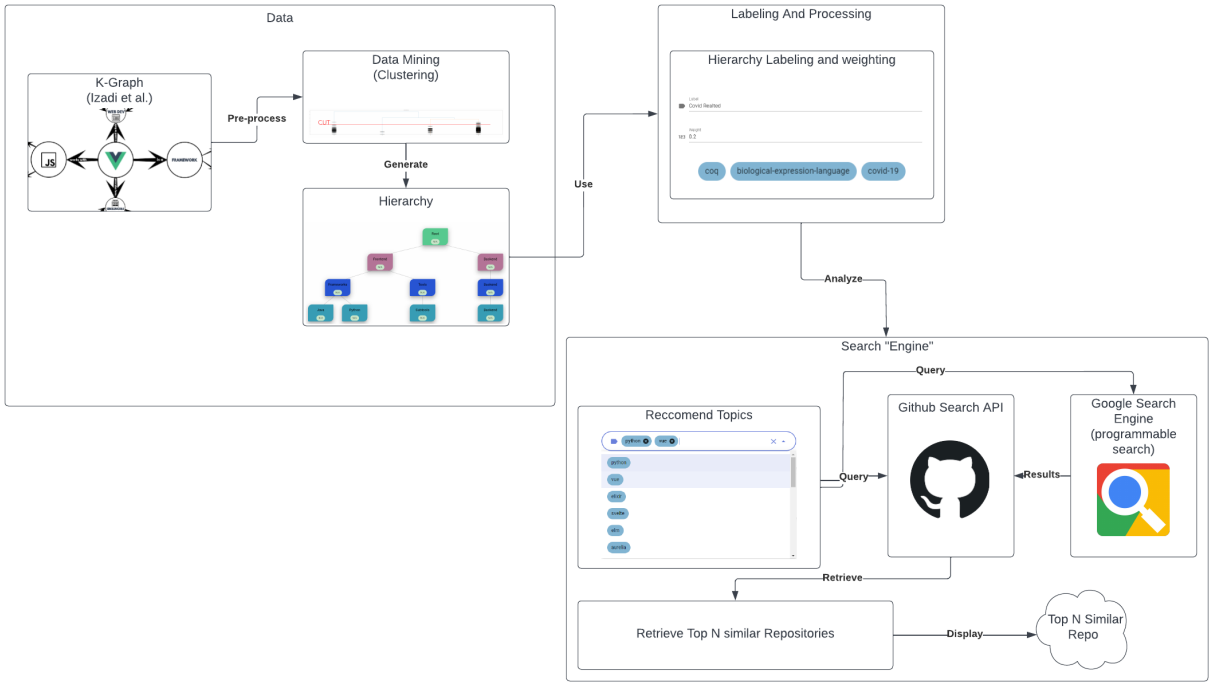

Figure 3: General Approach

each column represents a specific tag. For a tag T, the matrix M_T starts full of 0's entries, and when a relationship and tag are present for T, the value changes to 1. E.G., as seen in the list above, AWS will have in its representation the value 1 to the entry corresponding to the relationship $\xrightarrow{is-a}$ and the tag "Service".

Each tag representation is then clustered using clustering algorithms such as K-Means or Agglomerative Clustering.

4.1.3 K-Means Clustering

As seen in the Figure 4, the clusters obtained by applying the K-Means algorithm with the representation M (4.1.2) are well defined, but this is not enough since the K-means clustering does not produce a hierarchical structure.

4.1.4 Agglomerative Clustering

Agglomerative Clustering was used in order to create the automated hierarchy. As seen in the snapshot dendrogram in Figure 5, it can be observed that the hierarchical structure of this clustering technique, (e.g. material design, is the closest to bootstrap, and both are related to markdown). Based on such a hierarchical structure, we can make cuts, as seen in the Figure 5. Taking into account each cut, we merge the closest entries and continue with cuts until we reach the top of the dendrogram. Each cut step is stored locally, and after the cutting process ends, by using a recursive function, every cutting step is turned into a Hierarchy Level. This is done by taking into account the superior cuts, such that a current entry in the hierarchy is the child of a more extensive or equal cluster.

4.2 Hierarchy Visualisation

During the automatic hierarchy generation 4.1.4, it was concluded through expert evaluation that it is hard to manipulate and visualize such a hierarchy just based on textual information (such as the contents of the clusters). This is why we developed a fully-fledged web app[5] that helps with labelling, weighing and evaluating the hierarchy as seen in Figure 6.

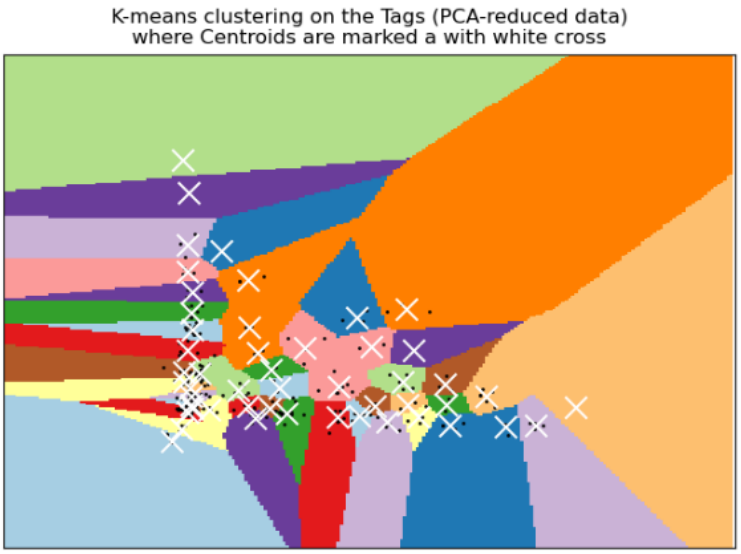

Figure 4: K-Means Clustering using the representation presented in 4.1.2

[5] https://hierarchy-visualiser.herokuapp.com/

4.3 Repository Recommender

This subsection will present the main approaches we used to recommend similar repositories to the user and the use of the Hierarchy 4.1.4 in the recommendation process. The first approach is based on the GitHub API, and the second is based on a wrap-around of the Google Search Engine and the GitHub API.

4.3.1 Hierarchy And Recommendations

To recommend repositories based on tags, the user needs to input the existing repository tags, and a piece of optional textual information about the repository's contents, since only 5% of all repositories available on GitHub contain topics/tags assigned to them [4]. A user is given an unordered list of available tags L = [x_1, x_2, x_3...x_n] that contains all the tags available in the hierarchy. When a tag is used in a search query, the list L is re-ordered based on the closest tags to the input one. Assume that list L_1 = [Vue, Python, Java, Angular, Ada]. According to the Hierarchy, Vue is closest to Angular, Python, Java, and Ada (in this order); the List L_1 will reorder all the entries basen on the closeness of the input tags to the rest of the tags.

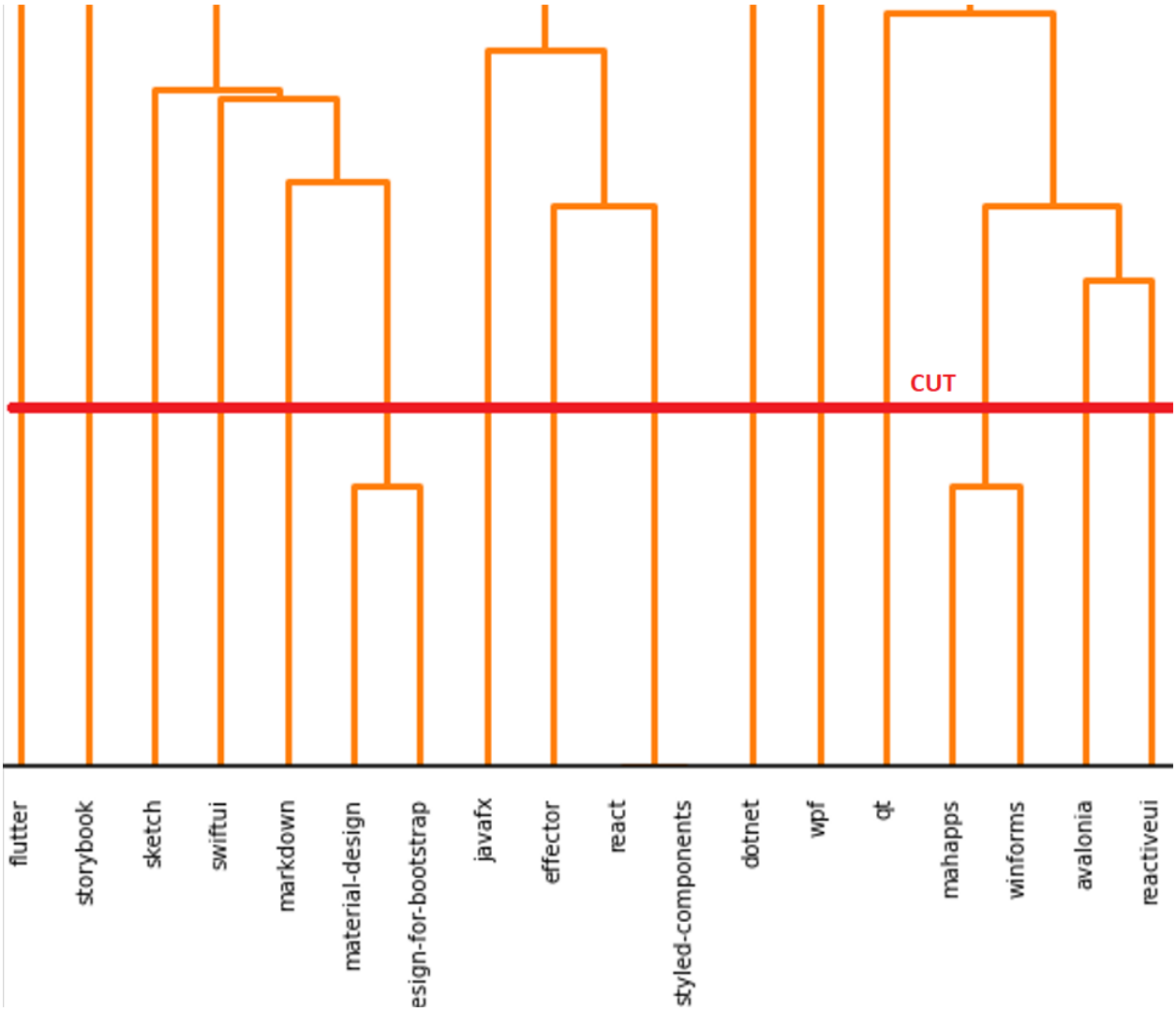

Figure 5: Agglomerative Clustering Dendrogram Snapshot With Highlighted Cut

4.3.2 GiHub Recommendations

To perform a similar repository recommendation using the GitHub API, the user needs to input the search query as presented in 4.3.1. Based on the user query, a GitHub-compliant query [8] is written and fed into the GitHub API in order to be consumed and provide similar repositories available in their open-source repository database. A query Q has the following form $Q = \{text : TI, tags : [t_1, t_2, t_3, ...]\}$ where 'TI' is the textual information provided by the user, and $t_1, t_2...$ are the tags chosen by the user using the hierarchy recommendation approach presented above. Each query Q is processed into a query where Q^* has the form $Q^*\{q =' TI + topic : t_1 + topic : t_2 + topic : t_3 + ...'\}$ and '+' represents a blank space. The query Q^* is fed into the GitHub API using the "Best Match" search option [9] to obtain an ordered list of similar repositories.

4.3.3 Google Recommendations

Similar to 4.3.2, a query Q^* is built based on a query Q. The only difference is that Q^* is used in a wrap-around Google Search Engine where $Q^* = \{q =' TI + t_1 + t_2 + t_3 + ...'\}$ no longer contains the qualifier '*topic* :'. Each query is then fed into the Google Programmable Search Engine [10] that acts as the wrapper around the Google Search Engine. The wrapper uses this search query to look for repositories on GitHub, the main open-source community repository host. Similar wrapping techniques are used in the field of code retrieval already [11, 12, 13], since a specialized search engine's algorithms currently include advanced indexing and ranking processes driven by the user interaction with data [14], making this strategy preferable to the traditional GitHub internal search. The Search Engine uses the following parameters:

- Search engine keywords (tuning keywords): repository
- Available in Site Restricted JSON API: ['github.com']
- Knowledge Graph Entities: ['Repository (version control)', 'Software Repository']

.

Each search request is fed using the Restricted JSON API available in the Google Programmable search. When a query is performed in the Google API, the results map all the information on Github.com, including discussions, issues and README files. Only the first six pages of results are considered in our approach since, through an organic search, more than 90% of the search users choose a result from the first page, almost 5% on the second page, and just 3% find good results on the pages 3, 4, 5 or 6 [15, 16]. When all the search results are gathered, they are filtered using the following rules:

- The link starts with "https://github.com"
- The link respects the GitHub repository path:
  "https://github.com/{AUTHOR}/{REPOSITORY_NAME}"
- The link does not contain in the {AUTHOR} placeholder the word 'topics' or 'blog'

After each result is filtered, the information for each repository is retrieved and validated by the GitHub API [1] verifying if the repository link maps to an existing repository and if the repository is available in an open-source form. The retrieved data from the GitHub API is sorted in the same order as the one provided by the Google Programmable search API, providing an ordered list of similar repositories.

5 Experimental Setup and Results

This section describes our experimental setup. We begin by stating the study's research goals, followed by explaining our hierarchy generation assessment but also the assessment of our similar repository recommender approaches. Following this, we present our baseline and the platform that facilitated the evaluation in contrast to the numerical results.

5.1 Research Question

How accurate and useful is a similar repository recommender built on the tags and the hierarchy?

5.1.1 Sub-Research Questions

- **RQ1:** How useful and accurate is a similar recommender repository based on tags?
- **RQ2:** How to identify and create a reliable hierarchy based on tags and the relationships between them? (e.g. Java $\xrightarrow{Framework}$ Spring $\xrightarrow{Testing}$ JUnit)
- **RQ3:** How to assess the performance when a hierarchy is used in a similar repository recommender system based on repository tags?
- **RQ4:** What search techniques are preferable when it comes to Similar Repository recommender?

The hierarchy assessment aims to address the RQ3 using two different types of development stages. In the first stage, we manually evaluated the quality of the clusters generated by the agglomerative clustering algorithm at the cuts 60, 50, 40, 30, 20, 10, 8, 6, 4, 3, 2, 1 with different methods such as "Ward", "Average" [17] and metrics such as "Euclidean", "Cosine"[18], as presented in the subsection 4.1.4. Based on the available results in the expert evaluation stage, we establish a simple yet efficient heuristic for concluding whether a hierarchy is good enough to be used in our similar repository recommender; more details will be presented in the results section 5.3.

5.1.2 Expert Evaluation

In the following list, we will explain the different approaches we analyse during the expert evaluation. All the items below can be visualized independently using our platform (hierarchy section)[6].

- Method: Single, Similarity: Cosine - The resulted hierarchy is an unbalanced tree. At the cut corresponding to 60 clusters, there exists a cluster that contains more than 90% of all the available tags. At the cut level 5 to 2, there are still singleton clusters. There is no need to analyze the contents of the clusters since more than 90% of the data results in the same cluster and does not provide any hierarchical information.
- Method: Single, Similarity: Cosine - Produces similar results as the one presented above. It does not provide hierarchical information since most of the data is condensed in one cluster.
- Method: Single, Similarity: Minkowski - Similar results as the others using the Single Method.
- Method: Complete, Similarity: Cosine - The resulted hierarchy is non-balanced; close to the root are singleton clusters, and there is a cluster around level 50 with more than 80% of the available tags.
- Method: Complete, Similarity: Euclidean - The resulted hierarchy is more balanced than the ones produced using the Single Method. There are no more singleton clusters close to the root, but there is still a cluster that contains more than 60% of the available tags at the cut level 60.
- Method: Complete, Similarity: Minkowski - The resulted hierarchy is similar to the one above that uses Euclidean similarity; the main difference between them is that around the cut level 60, there is a cluster that contains around 50% of the available tags. The hierarchy is more balanced than the ones above, and the contents of the clusters are also consistent compared with all of the above.
- Method: Average, Similarity: Euclidean, Minkowski - The resulted hierarchy is still unbalanced, with a node around the 50th cut that contains around 70% of all the available tags.
- Method: Average, Similarity: Cosine - The resulted hierarchy is more balanced than all the above methods and metrics. The data is evenly spread among the clusters, and it captures the connections between the topics (e.g. unreal-engine lands in the same cluster with the other engines and then merges into a bigger cluster that captures the "gaming tags" such as Twitch, Game-Jam)

[6] `https://hierarchy-visualiser.herokuapp.com/#/hierarchy`

- Method: Weighted, Similarity: Cosine - Provides similar results to the method above but misses the connection of some topics, e.g. AI with Image Processing or AI with synthetic biology.
- Method: Centroid, Similarity: Euclidean - Provides the most unbalanced hierarchy compared will all the methods above. It contains a cluster that includes almost all of the tags around the cut 50.
- Method: Median, Similarity: Euclidean - Produce a result almost identical to the Centroid-Euclidean method above.
- Method: Ward, Similarity: Euclidean - Provides the most balanced hierarchy compared with all the methods and metrics above. The hierarchy produced with these parameters keeps the qualities of the Average-Cosine method but also captures some information about the relationship between some topics (e.g. AI with Synthetic Biology)

5.1.3 Heuristics

Based on the data collected during the Expert Evaluation 5.1.2, we developed some heuristics that allow us to cross out some hierarchies without paying too much attention to the actual contents of each cluster inside the hierarchy.

- If the hierarchy is unbalanced, starting from the root to the bottom, and expert should analyse the contents of the closest clusters to the root. If the clusters are singleton clusters (the cluster's content contains just one tag), or if the clusters at a central cut (the middle level of the hierarchy) contain just a few tags compared with the other ones at the same level, cross out the hierarchy.
- A manual hierarchy should be created for certain tags so that, from an expert perspective, the items belong in the same category (cluster); for example, most front-end technologies should be merged in the same cluster. An expert should analyse the initial items of a hierarchy before applying any clustering algorithm to them. They should also verify the generated hierarchy with the manually created hierarchy, and see if the expectations are met. If the expectations are not met, they should try to analyze the generated clusters and discover if there are certain relationships between the tags that were not thought of; if there are, they should revise the manually created hierarchy and repeat the process. If the expectations are not met after a few iterations of this heuristic, the hierarchy must be crossed out.

5.2 Similar Repository Recommendation Assessment

To evaluate our Similar Repository Recommendation using the approaches presented in 4.3, we developed a fully-fledged web app[7] that implements the approaches and allows an exhaustive expert evaluation through a scenario-based evaluation [5].

5.2.1 Scenario Based Evaluation

The scenario-based evaluation consists of ten scenarios (Appendix A) that are analyzed by experts in the Computer Science field. Each expert is given one scenario at a time; they have to understand the scenario and write a query that depicts the needs illustrated by the scenario. Each scenario is represented in a textual form by a Software Engineering Project that needs to be implemented. All scenarios follow the same structure:

- Background about the person/company that needs the software.
- Background about the client's needs (e.g. soft/hard requirements) in a textual form without many technical details.
- The actual task for which the user has to write a search query that may lead to a Similar Repository that already implements the needed functionalities.

[7] `https://hierarchy-visualiser.herokuapp.com/#/evaluation`

Each user has to write a search query for each scenario (e.g. A.1) that is composed of a textual component and a recommended tag list (4.3.1) component as presented in 4.3. Each query produces three ordered lists that represent our two approaches presented in 4.3 and our baseline. The expert needs to analyse all the results and then choose the ones that are most relevant to the presented scenario.

5.2.2 Baseline

The baseline we are using to compare our approaches (4.3) is the classic GitHub "Best Match" Search Engine [9]. For the baseline, we use only the textual information compared with our two other approaches that also use a recommended tag list 4.3.1.

5.2.3 Web Platform

We created and launched an online web app to help with the expert review of the Recommender Systems. On top of the Hierarchy Visualisation feature presented in 4.2, we developed an evaluation system based on the Scenario-Based Evaluation procedure in 5.2.1. The Platform provides an initial overview of the tasks; then, it requests some data about the Experts that will start the evaluation process. When the information is collected, the expert begins the actual evaluation process. The specialist will receive one scenario and details about the steps needed to perform the evaluation:

- Analyse scenario.
- Write a query containing textual information and a tag list (the expert is instructed to focus more on the tags than on the textual information).
- Perform the query and analyse the results.
- Start the evaluation and select the relevant repositories to the scenario.
- Submit the evaluation.

When each scenario evaluation is submitted, the specialist will be presented with another scenario, until all the available scenarios were evaluated. Figure 7 represents the evaluation query search based on a scenario, Figure 8 depicts the search results based on the query, and Figure 9 depicts the repository selection from all of the three alternatives.

5.3 Results

In this subsection, we present the results of our approaches (4.3). Firstly, we will analyse the feedback received based on the scenario-based evaluation (5.2.1), using the evaluation platform we developed for this research paper (5.2.3) to answer the RQ1, RQ3 and RQ4. Secondly, we will look into the hierarchy assessment and present the results of using the hierarchy in our approaches to answer the RQ2. All the results below and also some other metrics can be live computed based on the current state of the feedback database and retrieved using our exposed JSON statistics endpoint[8].

5.3.1 Demographics

Firstly, we are looking into some information about the experts that evaluated our approaches (based on the user responses) using the scenario-based evaluation. In our study participated 14 people in total, as it can be seen in Table 1, where 71% were males, and 29% were women with an average age of around 18 - 22 years, as it can be seen in Table 2. 85% of the participants have a Bachelor's or equivalent in Computer Science related fields, as seen in Table 3. As it can be observed in Table 4, the average experience of the participants in Computer Science related fields is around 4 years.

[8] `https://similar-repo-backend.herokuapp.com/statistics`

Table 1: Gender distribution

Male	Female	Other	Total Participants
10	4	0	**14**

Table 2: Age Distribution

18 - 22	25 - 30	30 - 35
12	1	1

Table 3: Education Distribution

Bachelor's or equivalent	Master's or equivalent	Postdoctorate or equivalent
12	1	1

Table 4: Experience in Computer Science related field

$<$ 1 Year	1 - 2 Years	2 - 4 Years	4 - 6 Years	6 - 8 Years	8 - 10 Years	$>$ 10 Years
1	3	5	2	1	0	2

5.3.2 Scenario Results

This subsection first summarizes the overall statistics of the evaluation and then gives insights into the performance of our methods in a scenario-based manner. Table 5 summarizes the average completion time for each scenario A in minutes and seconds. The average completion time of the entire evaluation process takes around 63 minutes per evaluation expert. Furthermore, the accommodation time of the user with the platform is around 27 minutes, taking into account the time it takes on average for each user to evaluate the first two scenarios (A.1, A.2).

Table 5: Average Completion Time

Scenario	A.1	A.2	A.3	A.4	A.5	A.6	A.7	A.8	A.9	A.10
MIN:SEC	19:28	09:48	04:47	06:25	06:40	06:36	01:35	03:04	02:37	03:35

Table 6 summarizes the scenario metrics using main evaluation metrics MAP@3, P@3, and Re@3 (Recall@3) [19, 20]. The MAP@K metric is especially important for this type of recommendation system since our approaches provide a ranked list of similar repositories for each scenario. It may be observed in Table 6 that the first approach (GitHub) fails to outperform both the Google approach but also the Baseline in terms of all the metrics for all the scenarios. On the other hand, the Google approach outperforms the Baseline in terms of MAP@3 and Re@3.

Table 6: Scenario Metrics

Scenario	A.1	A.2	A.3	A.4	A.5	A.6	A.7	A.8	A.9	A.10
GitHub MAP@3	0.09	0.15	0.12	0.01	0.34	0.07	0.01	0.08	0.08	0.25
Google MAP@3	**0.44**	**0.42**	0.45	**0.34**	0.36	**0.36**	**0.50**	0.21	**0.45**	**0.46**
Baseline MAP@3	0.38	0.26	**0.58**	0.32	**0.51**	0.27	0.43	**0.39**	0.26	0.37
GitHub P@3	0.08	0.11	0.12	0.10	0.26	0.02	0.01	0.08	0.08	0.24
Google P@3	**0.43**	0.16	0.38	0.20	0.34	0.20	**0.48**	0.20	**0.37**	**0.46**
Baseline P@3	0.35	**0.21**	**0.56**	**0.60**	**0.40**	**0.30**	0.26	**0.35**	0.20	0.28
GitHub Re@3	0.25	0.23	0.15	0.00	0.38	0.07	0.08	0.08	0.08	0.30
Google Re@3	0.21	**0.54**	0.50	**0.50**	**0.50**	0.30	**0.58**	0.30	0.33	**0.60**
Baseline Re@3	**0.57**	0.53	**0.76**	0.49	0.61	**0.38**	0.50	**0.58**	**0.55**	0.41

Table 7 analyses the number of "hits" for each approach. A feedback result is considered a "hit" if our recommender system retrieves at least one relevant item. It can be observed that on average, the first scenario A.1 received more answers than the other scenarios. It may also be seen that the number of entries using the Google approach is lower than all the other approaches. An error caused this in the Google search engine, and the evaluation system invalidated some results. The Google Search Approach outperformed the GitHub approach and the baseline in the number of "hits". Furthermore, as in the previous metrics, the GitHub approach did not outperform any of the other approaches.

Table 7: Scenario Hits

Scenario	A.1	A.2	A.3	A.4	A.5	A.6	A.7	A.8	A.9	A.10
Hits GitHub	3	3	2	1	5	1	1	1	1	4
# entries GitHub	14	13	13	13	13	13	12	12	12	12
Hits Google	10	8	7	5	5	5	6	6	6	7
# entries Google	12	11	10	10	10	10	10	10	9	10
Hits Baseline	9	10	10	9	11	6	7	7	5	7
# entries Baseline	14	13	13	13	13	13	12	12	12	12

5.3.3 Approaches Results

This subsection will present the Overall approach results. Table 8 depicts our recommender system's overall statistics and metrics. The GitHub approach (4.3.2) provided no recommendations in 76% of the searches, the Google approach (4.3.3) and the Baseline provided recommendations in 81% and 80%, respectively, of the searches. On average, the Baseline outperformed the Google approach in terms of the number of recommended results but also in terms of the average number of relevant items. On the other hand, in terms of overall MAP@3 for all the approaches, the Google approach outperformed the Baseline approach by 18.75%. In terms of Recall@3, the Google approach outperformed the baseline by 16.50%. The Google and Baseline approaches outperformed the GitHub approach in all the metrics presented in Table 8.

Table 8: Search Approaches Overall Results

	GitHub	Google	Baseline
Average Recommendation Time	0.1 sec.	1.6 sec.	0.1 sec.
Empty Entities	76%	19%	20%
Non-Empty Entities	24%	81%	80%
Average number of results	2.4	9.6	11.9
Average number of relevant items	0.69	2.02	3.03
Proportion of relevant items	0.28	0.21	0.25
MAP@3	0.13	0.57	0.48
Overall Recall@3	0.11	0.49	0.42

6 Discussion

As seen in the demographics subsection 5.3.1, a total of 14 participants evaluated our similar repository recommender approaches. The average age of participants, the education distribution, and the experience in Computer Science Related Topics may impact the overall evaluation results. Our evaluation approach aims to evaluate the system using real-life scenarios, exposing the user to both familiar (e.g. A.1 - one-page website development) but also unfamiliar topics (e.g. A.6 - RIFD identification in an Airport) related to software development requirements, as received from a real-life client. We consider that the quality of the query is influenced by the time the user spends on the scenario and the overall experience of the user developing actual software. We believe that the understanding of the scenario and a good knowledge of the available technologies, influence the search for a similar repository.

Based on the results available in Table 5 (average completion time) and Table 6, we consider that the Google approach outperforms the Baseline approach when the user spends more time understanding the task in-depth.

We also observed that the GitHub approach could not be used in combination with tags. We noticed that the GitHub approach rarely returns any recommendation when more than one tag is used. We searched for more information about this behaviour and concluded that the GitHub search API could not search using tags properly. We assume that the GitHub Search API uses the "Best Match" search on the textual information and filters it with the tags. Since GitHub does not automatically add tags to repositories and there are a small number of repositories labelled with tags, the GitHub API cannot return any results.

We consider that even though the Google approach performs the best compared with the other approaches, it cannot be used in a free application. Since the free Google Search cannot be used or scrapped without launching the "reCAPTCHA" system, Google's programmable search needs to be used instead (paid service). During our evaluation and validation of the data, we reached an average cost of 1.7 euros for each evaluation process (per user).

7 Responsible Research

In this section, we examine the potential challenges to the study's validity, how we overcame these concerns, and the reproducibility of the methods we used during our study regarding possible validity issues.

7.1 Internal Validity

These types of threats correspond to the correctness of the ground truth of relevant items chosen by the experts during the evaluation procedure, but also the subjectiveness of the users for each evaluation scenario. We address this problem by giving the evaluators specific guidelines before starting the evaluation task and during the task (availability of the guidelines during the evaluation process and instructions that are displayed directly in the scenario - e.g. A.1). The subjectiveness is reduced by manually sampling the results and checking if the ground truth respects the guidelines. If it does not, the response is invalidated.

Another threat may be the failure of the external services we are using in our approaches (e.g. Google Programmable Search). To solve this issue, we implemented an automated "flagging" system that invalidates the results if the external dependencies fail or are unavailable during the evaluation.

7.2 External Validity

These types of threats correspond to the generability and effectiveness of our approaches. There are cases when our system cannot provide proper recommendations since the input data does not correspond with any available open-source repository (e.g. a specific quantum-computing algorithm) at a particular point in time. We solved this by developing our approaches in such a way that, at some point in time, when an open-source repository that corresponds with the search query will be available, the recommender system would be able to recommend it (the recommendation model is not static).

7.3 Construct Validity

These types of threats correspond with the capabilities of our fully-fledged application (5.2.3) that is used to generate and visualise hierarchies but also to recommend and evaluate similar repositories. The contents of the hierarchy may change over time according to the open-source community trends; this also means that the user approaches should be updated based on the new trends or data. Due to this data volatility, our system allows the hierarchical data to be changed in an accessible way by updating the K-Graph used for clustering (4.1.4). Furthermore, the approaches automatically adapt (as long as the hierarchy keeps the same format) to the newly created hierarchy, due to the Front-End Back-End architecture. The whole web app was developed using a "component structure", where every Graphical User Interface (GUI) component is reusable (when one component changes its structure/functionality, the functionality will be available on every page that is using the component automatically).

7.4 Reproductibility

To ensure that this paper can be easily reproduced, we provide two replication packages containing all the data we used for our study and the existing frameworks/tools we used to answer the research questions. The first replication package[9] contains the front-end application that allows to reproduce the data visualisation presented in section 4.2, the search methods we considering during our research (4.3.2, 4.3.3) but also the evaluation framework we developed for this study presented in subsection 5.2.1. The second replication package[10] contains the hierarchy builder presented in subsection 4.1.4, the statistics (evaluation metrics) presented in subsection 5.3 and the Google Search approach endpoints.

8 Conclusions and Future Work

This paper analyses whether a similar repository recommender system built on tags and a tag hierarchy can be created and used reliably. We also analyse how a tag hierarchy can be established, evaluated and used in such a recommender system. Furthermore, we looked into how to evaluate such a recommender system but also what search approaches are preferred by the experts when searching for similar repositories.

The results show that a similar repository recommender system based on tags and a hierarchy is valuable and accurate from an expert perspective. The results also show that a tag hierarchy can be used to recommend similar tags to the users during the similar repository search procedure. During the study, we developed and deployed a fully-fledged framework (web-app 5.2.3) that allows us to create, label, weight, and evaluate any tag hierarchy (both manually created or automatically created). Based on the results, we concluded that the hierarchy provides reliable recommendations that improve the search for similar repositories.

During our study, we established some heuristics that allow us to assess the performance of any hierarchy in combination with the web app. Moreover, we developed and deployed a scenario-based evaluation framework that allows the use of the hierarchy in searching for similar repositories using the Google (4.3.3) and GitHub (4.3.2) search approaches in comparison with our Baseline. We concluded, based on the evaluation results, that the Google Search approach is prefered over the GitHub approach both from an accuracy perspective and also from a user perspective. The Google Search approach outperformed the Baseline by 18.75%, and the GitHub Search by more than 100%.

We published the study replication package both for the front-end[9] (data visualisation, evaluation, approaches) and the back-end[10] (hierarchy generator, statistics, approaches).

In the future, we will focus on introducing the hierarchy as a new recommendation layer for filtering and reordering the recommendation results. Furthermore, we want to study the performance of a similar repository recommender system using more search engines such as Bing or DuckDuckGo and explore the implementation of the Google Knowledge Graph [21] in our Google approach (4.3.3).

[9] `https://github.com/andrei5090/SimilarRepoRecommender/tree/main/webapp`
[10] `https://github.com/andrei5090/SimilarRepoBackend`

References

[1] Github, *Classifying your repository with topics*, 2021. [Online]. Available: https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/classifying-your-repository-with-topics

[2] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. Di Penta, and M. Lanza, "Supporting Software Developers with a Holistic Recommender System," *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017.

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

[4] M. Izadi, A. Heydarnoori, and G. Gousios, "Topic recommendation for software repositories using multi-label classification algorithms," *Empirical Software Engineering*, vol. 26, no. 5, 2021.

[5] N. Looker, D. Webster, D. Russell, and J. Xu, "Scenario Based Evaluation," *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008.

[6] A. H. Maliheh Izadi, Mahtab Nejati, "Semantically-enhanced Topic Recommendation Systems for Software Projects," *Empirical Software Engineering*, vol. 26, no. 5, 2021.

[7] F. Lopez de la Mora and S. Nadi, "Which library should i use?: A metric-based comparison of software libraries," in *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, 2018, pp. 37–40.

[8] GitHub Docs, *Understanding the search syntax*, 2022. [Online]. Available: https://docs.github.com/en/search-github/getting-started-with-searching-on-github/understanding-the-search-syntax

[9] GitHub, *GitHub Search*, 2022. [Online]. Available: https://docs.github.com/en/rest/search

[10] Google, *What is Programmable Search Engine? - Programmable Search Engine Help*. [Online]. Available: https://support.google.com/programmable-search/answer/4513751?hl=en

[11] C. T. Brock Angus Campbell, "NLP2Code: Code Snippet Content Assist via Natural Language Tasks," Tech. Rep., 01 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1701.05648

[12] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," Tech. Rep., 2013.

[13] F. F. Xu, B. Vasilescu, and G. Neubig, "In-ide code generation from natural language: Promise and challenges," 2021. [Online]. Available: https://arxiv.org/abs/2101.11149

[14] Y. Wei, N. Chandrasekaran, S. Gulwani, and Y. Hamadi, "Building bing developer assistant," Tech. Rep., May 2015.

[15] Chitika Insights, "The Value of Google Result Positioning," Tech. Rep., 02 2022. [Online]. Available: https://research.chitika.com/wp-content/uploads/2022/02/chitikainsights-valueofgoogleresultspositioning.pdf

[16] P. Petrescu, "Google Organic Click-Through Rates in 2014," 03 2021. [Online]. Available: https://moz.com/blog/google-organic-click-through-rates-in-2014

[17] SciPy, *scipy.cluster.hierarchy.linkage SciPy v1.8.1 Manual*, 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage

[18] SciPy , *scipy.spatial.distance.pdist SciPy v1.8.1 Manual*, 2021. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html#scipy.spatial.distance.pdist

[19] R. J. Tan, "Breaking Down Mean Average Precision (mAP) - Towards Data Science," 03 2022. [Online]. Available: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52

[20] J. Arguello, "Evaluation Metrics," 2013. [Online]. Available: https://ils.unc.edu/courses/2013_spring/inls509_001/lectures/10-EvaluationMetrics.pdf

[21] A. Singhal, "Introducing the Knowledge Graph: things, not strings," 05 2012. [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/

A Evaluation Scenarios

A.1 Scenario Task 1

A car competition organiser wants to promote their business. They want to build a website about their available competitions. The website is just an informational one; the users will not be able to reserve/buy any tickets for the competitions. The website will follow a parallax scrolling style (1-page website) with information about the competitions, such as location and entry fee. The information on the website needs to be updated just once every few years when the organisers adjust the prices with the inflation. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.2 Scenario Task 2

A small chair manufacturing business wants to minimise its "bookkeeping" costs. Every month, all the bills regarding materials or energy are centralised in their bookkeeping system by the business owner. The business owner thinks that he has more important things to do but doesn't afford to hire another person to centralise the bills. All the bills are received at the address: accounting@nicechairs.com. The business owner wants to automate this bookkeeping process. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.3 Scenario Task 3

An NGO from Guatemala wants to promote their country worldwide. They plan to build a mobile application that allows users to plan their journey in Guatemala. The application should allow the users to choose their interest points and plan a trip, including transport, tickets, and accommodation. The app aims to plan the trip so that it optimises the route to the touristic objectives and optimises the waiting time at the interest points (e.g. a museum). Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.4 Scenario Task 4

Jerry just won the lottery and decided to open a bar in his hometown. Last year, he went on a trip to the USA and saw live augmented reality Pool tables that can track your actions and recommend the best shots projecting a live animation on the table. Even though Jerry won the big jackpot, he doesn't want to invest much money in such a table; the starting price on the market is 15000 euros. He already bought the Pool table but needs somebody to build this augmentation system. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.5 Scenario Task 5

The mighty food vlogger "ImiPlaceSaMananc" travels worldwide to find the best shawarma on the planet. He keeps a digital journal about the characteristics of each shawarma he tasted, the ingredients, and the restaurant's location that prepared the food. At the end of each tasting, he writes down the "food grade" (e.g. Above Average, Average...etc.). He wants to share his Shawarma tasting skills with the world. This is why he wants to build a recommender system that recommends the best shawarma in the user's area (based on his journal). Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual

search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.6 Scenario Task 6

The Eindhoven airport wants to create an embedded system that is able to track the RIFD chip in your passport at the self-check-in gate, so you don't have to take it out on your passport and scan it. They want each self-check-in gate to work independently. When you come close to the gate, it will automatically recognise your passport and start scanning your biometrics (the system will provide a video stream of the user's face or fingerprints). The biometric validation is not a concern at the moment since another service provider will handle this. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.7 Scenario Task 7

The TU Delft Discord server admin wants to automate the filtering system. He wants to filter all the messages that contain swear words or graphic pictures that violate the TU Delft standards. This is why he wants to create a filtering system that can emit "warnings" or server "bans" according to specific criteria. The filter will act as a discord bot that scans all the messages and take action when required. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.8 Scenario Task 8

A big tech company wants to provide bikes to their visitors and employees. Each bike has a card scanner attached to it. A visitor or employee card needs to be scanned to unlock a bike. The bikes can be unlocked with a card, but the administrators have no information on how many bikes are used at certain timestamps or the location of each bike. This is why they need a system to centralise the data of each bike and display information such as free bike locations in a GUI. They have no preference for how the system is implemented. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.9 Scenario Task 9

WorkAholic INC. wants to implement a task prioritisation system in their workflow. At the moment, the task difficulty and time needed to finish the Team Managers establish it. The CTO observed that the managers sometimes give shorter task times to please their superiors. This is why the WorkAholic INC. wants to implement an automated task "assessor" that suggests the work time needed for a task to be solved and the assignment's difficulty. Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

A.10 Scenario Task 10

A small company specialising in HR management wants to improve their API performance. They provide an API to their clients that allow them to input their HR data and provide metrics and statistics about employees' performance. After an internal audit, they observed that some metrics and statistics take too much time and resources to compute, but the results are relatively constant for most of their clients. They want to solve this issue cost-effectively since this performance issue has already severely affected their income.

Please provide a query that may lead to an already built solution for this type of application or a stack of technologies that may help solve this task. Please focus more on the tags than on the textual search. Please keep in mind that the order of the tags is changing and provides you with recommendations when you input at least one tag.

B Platform

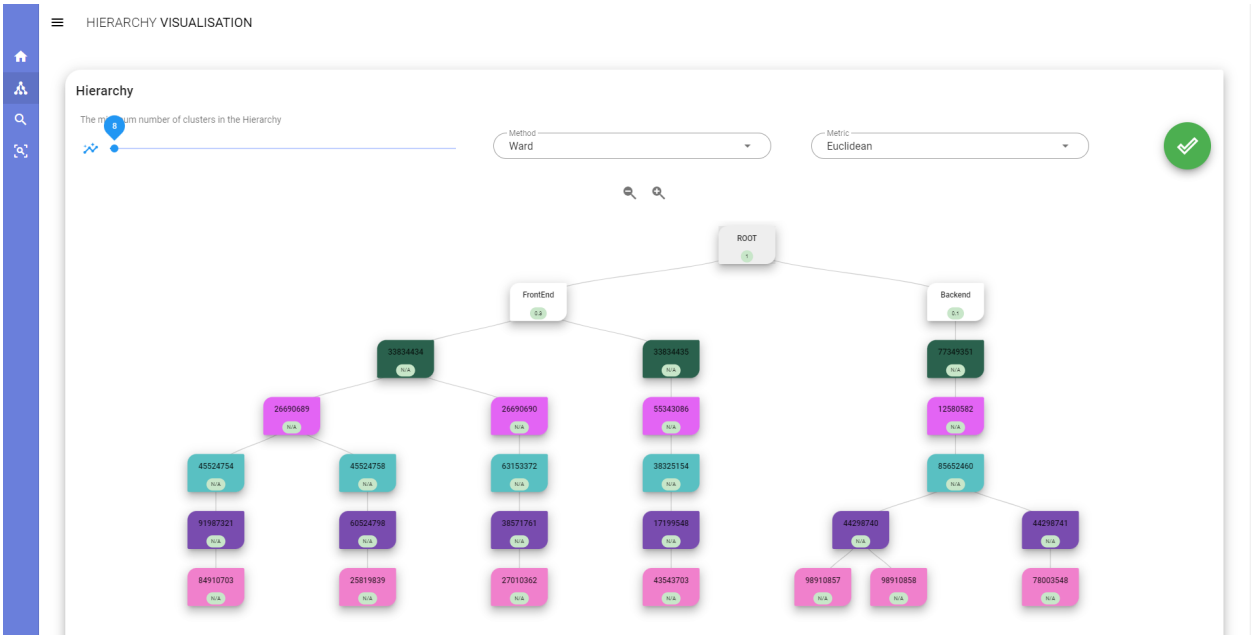

Figure 6: Hierarchy Visualiser WebApp

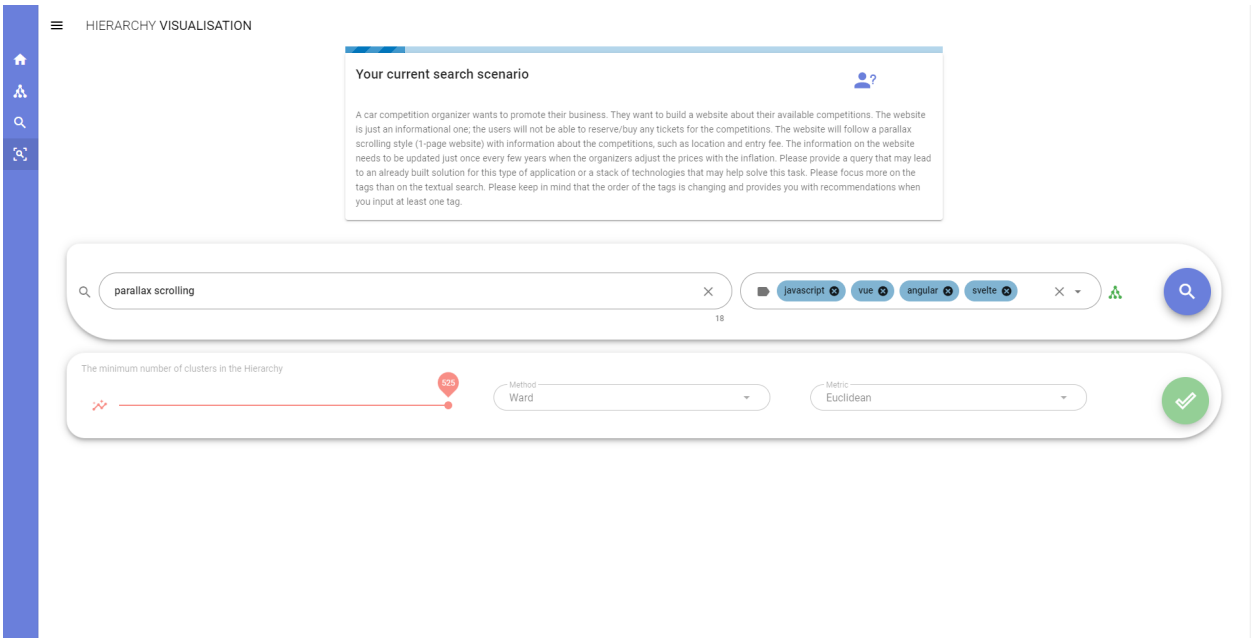

Figure 7: Evaluation Query Search

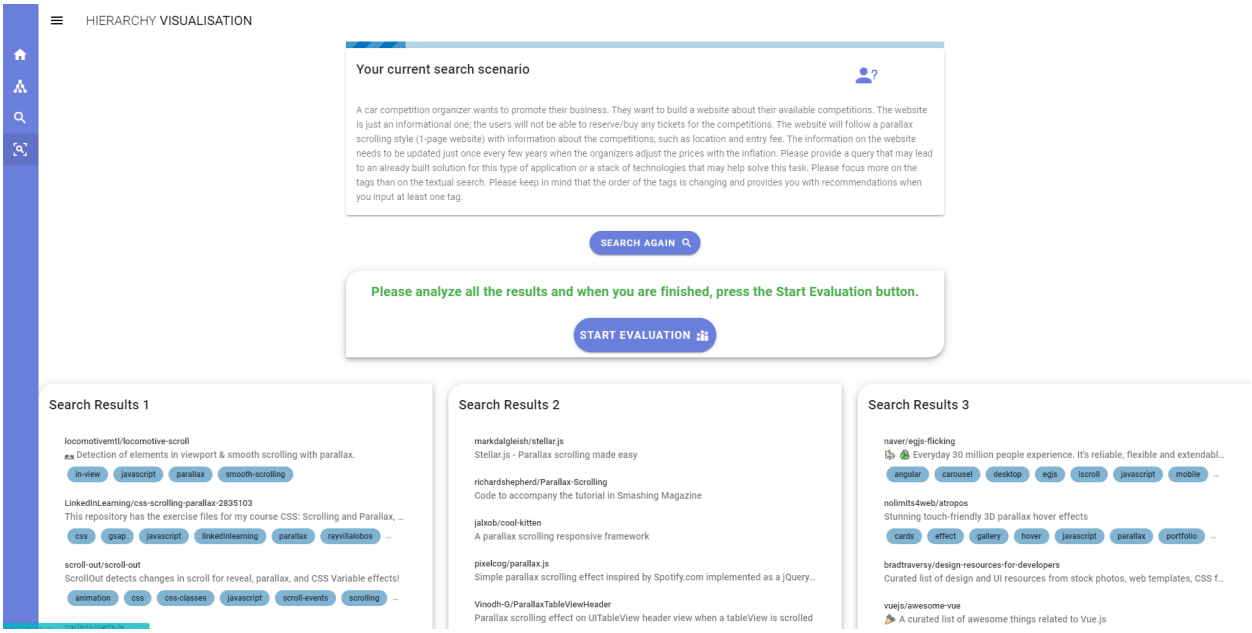

Figure 8: Evaluation Query Search Results

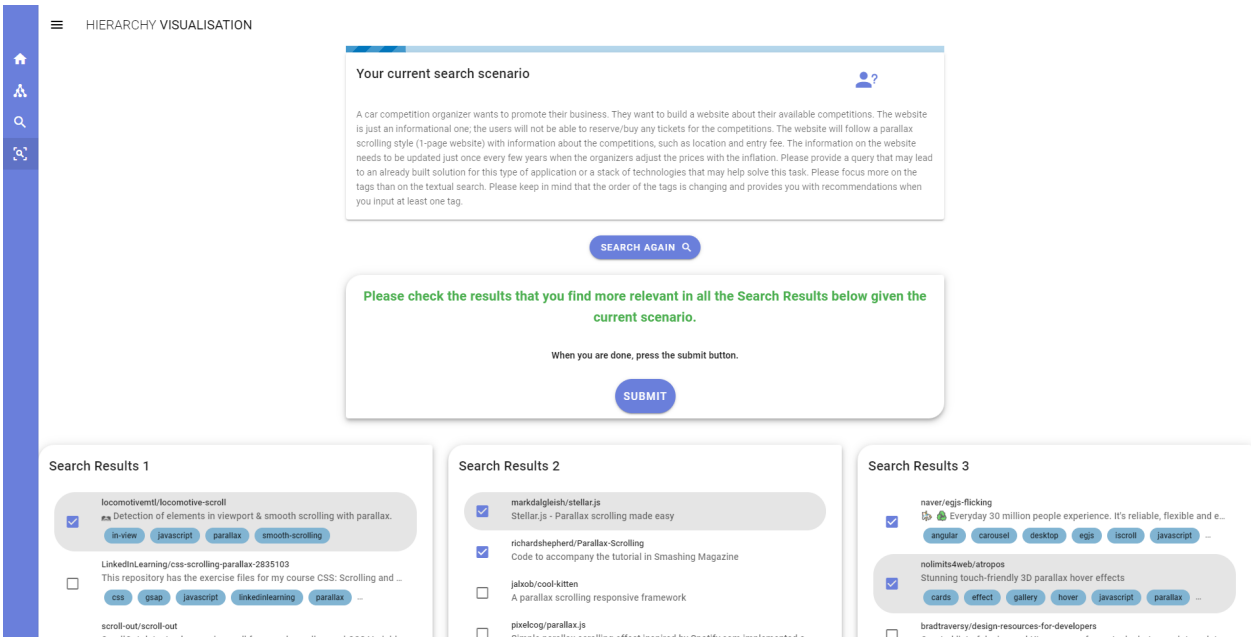

Figure 9: Evaluation Repository Selection