Journal of Computer Languages

AI-based Clustering of Similar Issues in GitHub's Repositories

--Manuscript Draft--

Manuscript Number:	COLA-D-23-00096
Full Title:	AI-based Clustering of Similar Issues in GitHub's Repositories
Article Type:	Full Length Article
Keywords:	Similar Issues; GitHub; Machine Learning; Maintenance; Clustering
Corresponding Author:	Hamzeh Eyal Salman Mutah University Karak, JORDAN
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Mutah University
Corresponding Author's Secondary Institution:	
First Author:	Hamzeh Eyal Salman
First Author Secondary Information:	
Order of Authors:	Hamzeh Eyal Salman
Order of Authors Secondary Information:	
Abstract:	Issues on GitHub are submitted to the issue tracking system for different maintenance activities. The attractive repositories receive a large number of issues daily. Assigning similar issues individually to different developers for validating and fixing introduces inconsistencies and slowing the fixing process.. However, grouping similar issues into clusters and assigning each cluster to the same developer/team speeds up the fixing process. A machine learning algorithm-based approach has been proposed to group similar issues together. The approach was applied to 13 software components from large repositories. Findings reveal that the proposed approach identifies similar clusters of issues with promising results.
Suggested Reviewers:	Zakarea Alshraa zmalshara@just.edu.jo
	Abdelhak-Djamel Seriai seriai@lirmm.fr
	Marianne Huchard huchard@lirmm.fr
Opposed Reviewers:	
Additional Information:	
Question	**Response**
Free Preprint Service Do you want to share your research early as a preprint? Preprints allow for open access to and citations of your research prior to publication. Journal of Computer Languages offers a free service to post your paper on SSRN,	NO, I don't want to share my research early and openly as a preprint.

an open access research repository, when your paper enters peer review. Once on SSRN, your paper will benefit from early registration with a DOI and early dissemination that facilitates collaboration and early citations. It will be available free to read regardless of the publication decision made by the journal. This will have no effect on the editorial process or outcome with the journal. Please consult the SSRN Terms of Use and FAQs.	
To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF). (If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.)	Data will be made available on request.
To complete your submission you must select a statement which best reflects the availability of your research data/code. IMPORTANT: this statement will be published alongside your article. If you have selected "Other", the explanation text will be published verbatim in your article (online and in the PDF). (If you have not shared data/code and wish to do so, you can still return to Attach Files. Sharing or referencing research data and code helps other researchers to evaluate your findings, and increases trust in your article. Find a list of supported data repositories in Author Resources, including the free-to-use multidisciplinary open Mendeley Data Repository.)	Data will be made available on request.

AI-based Clustering of Similar Issues in GitHub's Repositories

Hamzeh Eyal Salman[a,*]

[a]*Department of Software Engineering, Mutah University, 61710 Al-Karak, Jordan*

Abstract

Issues are highly prevalent on GitHub due to the increasing scale of its software repositories. These issues are submitted to the issue tracking system for several reasons: reporting a bug, asking a question, or other maintenance activities. The attractive repositories on Github receive a large number of issues daily. Assigning similar issues individually to different developers for validating and fixing introduces inconsistencies when asynchronously independent developers fix them, in addition to slowing the fixing process. However, grouping similar issues into clusters and assigning each cluster to the same and appropriate developer/team speeds up the fixing process. In this paper, a machine learning algorithm-based approach has been proposed to support issue management on GitHub by grouping similar issues together. For validity, the proposed approach was applied to 13 software components from different and large repositories. Findings reveal that the proposed approach identifies similar clusters of issues with promising results using widely used evaluation measures in this subject: Precision, Recall, and F-measure.

Keywords: Issues, Issues Management, Similarity, GitHub, Machine Learning, Maintenance, Clustering

1. Introduction

In the software development life cycle, the maintenance phase is crucial for several reasons that form the motivation for this phase. Firstly, maintenance activities accommodate the users' demands for improvement and extension via adding new features or modifying the existing ones. Secondly, the source code should be free from any potential flaw that badly impacts the performance or correctness [1]. Finally, maintenance activities help facilitate future maintenance efforts such that the maintainer spends a little time and effort for understanding software artifacts to achieve different maintenance activities with the least cost [1].

Maintenance efforts on GitHub, the most popular social coding platform, escalated. This is because it hosts a huge number of repositories with a huge number of contributors, more than 100 million developers working on over 352 million repositories in November 2022 [2][3]. On GitHub, a contributor creates issues in a repository for several maintenance reasons [4]: adding new features, reporting bugs to be fixed, or asking a question about the capability of the software. The core team members of the repository should respond to these issues to keep the contributors engaged in the improvement process of their software project. However, when the software project grows, the number of contributors and thus the number of issues grows incrementally, too. For example, *Elastic-search* project receives daily and monthly 25 and 760 issues, respectively [5]. These issues differ in their nature and quality (asking for support, for improvement of a functionality, bug reporting). Therefore, the issues management in such cases will be harder.

Assigning similar issues individually to different developers for validating and fixing introduces inconsistencies when asynchronously independent developers fix them, in addition to slowing the fixing process [6]. However, grouping similar issues together into clusters and assigning each cluster to the same and appropriate developer/team speeds up the fixing process. In this study, I suggest an approach to support similar issues clustering on GitHub. This approach aims to group similar issues into clusters. Such a clustering

*Corresponding author
Email address: hamzehmu@mutah.edu.jo (Hamzeh Eyal Salman)

mechanism has positive impacts on issue management and solves different problems as follows: (i) to speed up the process of fixing issues and reduce the turnaround time for fixing, (ii) when related issues are grouped together, this speeds up the assignment process by finding the appropriate reviewer(s) for a cluster instead of individual issues, especially, if the assignment process is done manually [6], (iii) to consolidate similar issues into a single, well-defined issue that covers all the variations and aspects of the problem. To the best of my knowledge, clustering similar issues on GitHub is seldom considered in the literature, and many approaches were proposed to identify only duplicate issues. The difference between similar and duplicate issues is that the former refers to a set of issues with some common characteristics or are closely related, but they are not necessarily exact duplicates. For example, if users are encountering slightly different issues while using the same feature of a software application, these issues might be considered similar. The latter refers to copies of the same issue submitted by different users with/without the same descriptions.

The issue tracking system (ITS) in GitHub is an important maintenance tool to manage issues. This tracking system allows contributors to provide descriptive information for issues such as *title* and *description*. Also, it is working as a labeling system. Labels are meta-data assigned to issues for different purposes. Namely, labels give information about the topic of issues, their priority, their objectives, etc. Moreover, they support core team members of the project in classifying and triaging. Liao et al. [7] studied the impact of labeled or tagged issues on their management using six popular repositories. They found that the labeled issues were immediately addressed while unlabeled ones could be opened for a long time. However, labeling issues in GitHub are scarcely used and sometimes with wrong labels [8][9]. Therefore, issue labels can not be trusted to assign issues to reviewers.

The proposed approach takes, as input, issues from a given repository. Then, it computes the textual similarity among these issues. This similarity is computed based on the textual information, which includes the *title* and *description* of each issue. Such information embeds the intent and object of the contributor who submits the issue. This textual information represents features to help machine learning algorithms to cluster similar issues together. Next, the Agglomerative Hierarchical Clustering (AHC) algorithm is used to cluster similar issues. To evaluate the efficiency of the proposed approach, the obtained results are evaluated using the well-known metrics in this subject: Precision, Recall, and F-Measure.

The remaining section of this article is structured as follows. In Section 2, a detailed background is presented to understand the proposed approach. In Section 3, a literature review related to the research work of this article is surveyed. The proposed approach is detailed in Section 4 followed by the experimental results and discussion in Section 5. Finally, conclusions and future work are presented in Section 6.

2. Background

In this section, a necessary background to understand the issue report structure on GitGub, issues management through the issue tracking system, and agglomerative hierarchical clustering algorithm is presented. Also, a motivational example is provided in this section to support this research work.

2.1. GitHub

Nowadays, social coding platforms attract developers and end users as these platforms provide a huge number of free open-source projects. One of the most famous social coding platforms is GitHub. In November 2022, there are more than 100 million developers working on over 352 million repositories [2][3]. GitHub is established on top of the Git version control system. Therefore, it provides different development and maintenance activities such as parallel development and integration, repository management, issue reporting, tracking, and fixing, etc. In GitHub, there are two main social distinct artifacts: pull-request (PR) and issue reporting. These artifacts are produced by two different processes: development and testing. The issue is used by testers to inform a repository's owner(s) about discovering a buggy code or any other type of issues and how to generate them. PR is used by developers or maintainers to merge their code changes to the main branch of a given repository. These code changes may fix an issue, add feature(s), enhance performance, or any other development activity [10]. Therefore, issues usually are linked to PRs to show their fixing progress in GitHub.

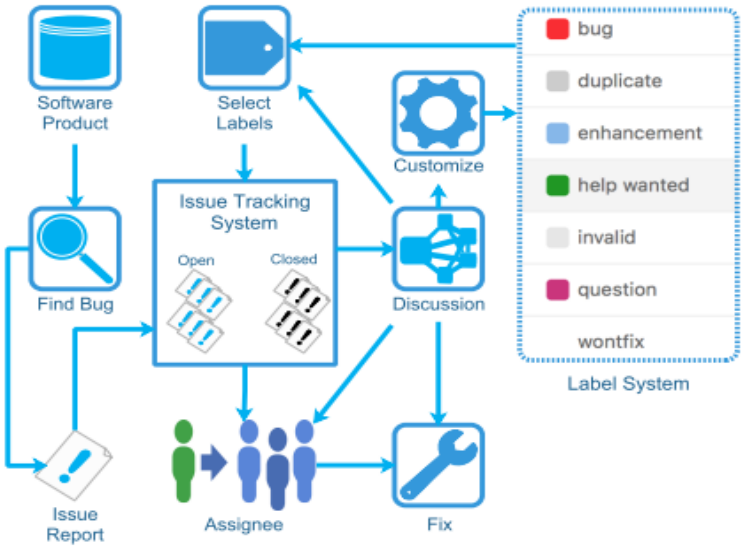

Figure 1: An overview of Issue Tracking System in GitHub [4].

The issue tracking system (ITS) provided by GitHub is a lightweight web-based application to manage and maintain a list of issues and track their progress. Figure 1 shows an overview of the workflow of the issue tracking system in GitHub [4]. This overview consists of the following steps. Firstly, as shown, when a tester discovers an issue in a particular repository, he opens or creates an issue report to report this issue via ITS. It typically asks him to provide elementary information to describe this issue in addition to other structured fields that should be filled in to submit such a report. Secondly, to fully grasp the issue, the repository's owner(s) and contributor(s) have a discussion about it after receiving the issue report. Thirdly, this issue is labeled with meta-data such as its category, priority, etc. Finally, based on this labeled data, the issue is assigned to the appropriate developer to fix.

Each issue on GitHub has a title, description, number, and status (either closed or open). The title and description include implicit useful information about the issue's objectives and context. The description may include a code snippet to refer to and detail the buggy code. Also, each issue has other different information such as labels, comments, assignees, contributors, milestones, etc. Issues may be linked to pull requests that resolve them. Labels give different important information about the issue type (bug), the severity (inconvenient), and others. All these labels are important for issue management.

2.2. Motivational Example

Four similar issues are presented in Figure 2 to motivate the need to cluster these similar issues together. These issues[1],[2],[3], [4] are selected from *Scrapy* repository on GitHub.

As shown in this figure, there are four issues with similar titles and descriptions. The titles of these issues indicate that they are about cookie handling in the *Scrapy* project. Also, the description of these issues contained shared keywords, such as spider, cookie, handling, middleware, and request. By referring to the description of these issues, it is noticed that they are related and cover different aspects of cookies in *Scrapy*. Moreover, the description explicitly indicates that these issues are related (see blue boxes in the figure). For example, issue number 5431 indicates that this issue is related to issue numbers 2124 and 5841, respectively.

[1]https://github.com/scrapy/scrapy/issues/5930
[2]https://github.com/scrapy/scrapy/issues/5431
[3]https://github.com/scrapy/scrapy/issues/2124
[4]https://github.com/scrapy/scrapy/issues/5841

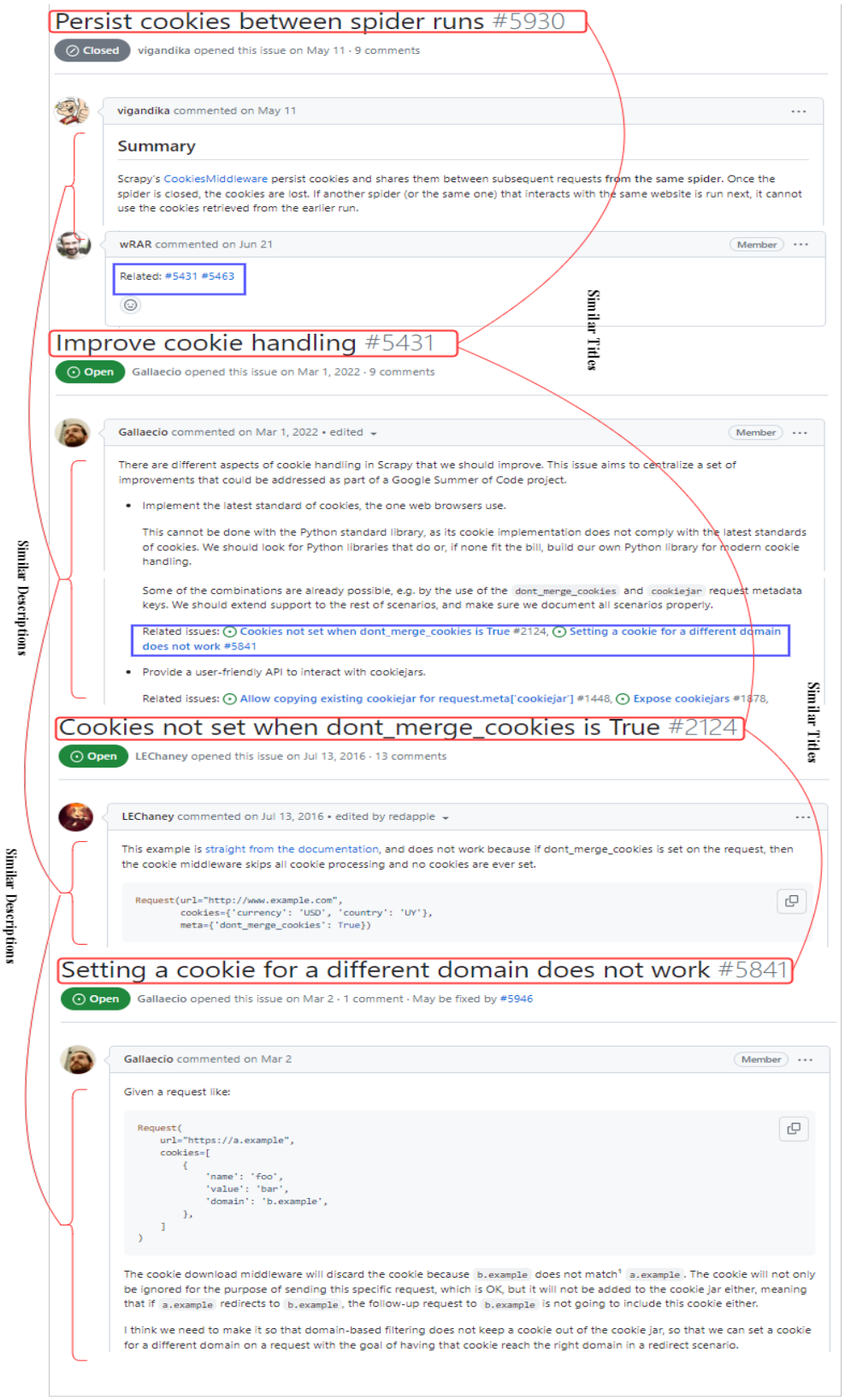

Figure 2: An example of four similar issues from *Scrapy* repository on Github.

2.3. Clustering Algorithm

Clustering is an essential activity in unsupervised machine learning (ML) [11]. Generally, clustering aims to find similar unlabeled data instances and group similar instances into groups. Each group is called a cluster. Such similarity is computed based on features embedded in the data. In the literature, there are different clustering algorithms such as K-Means, BIRCH, DBSCAN, agglomerative hierarchical clustering (AHC), etc. The choice of which clustering algorithm should be applied depends mainly on two reasons. Firstly, the number of clusters that should be defined in prior. For example, the applications of K-Means and BIRCH require a predefined number of clusters to be known, while DBSCAN and AHC do not. The second one is the need to build a generalized prediction model for incoming future unlabeled data instances. For example, K-Means and BIRCH allow building prediction models for future use, while DBSCAN and AHC do not allow building such models. All clustering algorithms require ground truth clusters to be available prior to evaluate their efficiency. In this study, the AHC algorithm is used as there is no predefined number of similar issue clusters, and thus it is fit for the purpose of this research work.

The AHC algorithm is a bottom-up hierarchical clustering where each individual data instance forms a single cluster [12]. The nearest data instances according to some similarity measure (either semantic or lexical similarity measure) are merged together to form a new cluster. This initial binary merging of data instances and later of emerging clusters continues until all data instances in the problem space belong to only one cluster, and the resulting hierarchy called *dendrogram* tree. Finally, this tree structure is cut horizontally to produce a specific number of clusters. The cutting point is driven by a distance threshold or by other methods, such as *Elbow* method. When the cutting point goes up or down in the dendrogram tree, the number of classes decreases or increases, respectively.

3. Related Work

To the best of our knowledge, there is no research work in the literature addressing the problem of clustering similar issues into groups in GitHub. In the literature, there are many approaches proposed to discover duplicate issues. These approaches are mainly organized into three categories based on the methodology being used: natural language processing (NLP), information retrieval (IR), and machine learning (ML). However, NLP and IR-based approaches are limited to detecting duplicate issues due to textual and vocabulary mismatches in the text describing these issues [13][14]. Also, ML-based approaches suffer from a lack of large and quality data, generalized prediction model, and data imbalance [13]. In this section, the most recent and relevant works in these categories are presented.

3.1. IR and NLP-based Approaches

The idea behind using IR and NLP is to find textual overlapping between bug report queries and already existing report queries in the repository. In [15], Runeson et al. built a vector space model (VSM) of bug report textual information (summary and description fields) to detect duplicate bug reports. Then, different similarity measures are used to find the similarity between the query bug reports and the already existing bug reports: Jacard, cosine, and dice measure. In [16], Wang et al. improved the work proposed by Runeson et al. by considering execution traces available in bug reports in addition to the textual information. However, these traces are not always available in bug reports. In [17], Amoui et al. proposed to use IR to retrieve a ranked list of top-k similarity bug reports in an industrial use case (BlackBerry Systems). The ranked list is computed based on three features extracted from the report of these systems: Summary, Description, and First_Email. This ranked list allows triggers to compare the incoming bug report with this list to discover the duplication. In [18], Thung et al. used VSM to find duplicate bug reports based on the title and description of bug reports.

Topic-based modeling has an important impact on detecting duplicate bug reports. In [19], Nguyen et al. proposed to combine IR and topic models (semantic information in bug reports) to detect duplicate bug reports. In [20], Akilan et al. proposed to combine Latent Dirichlet Allocation (LDA) and word embedding technique (called GloVe) to detect duplicate bug reports. In [21], Hindle et al. calculated the similarity between the title, description, product feature, component feature, bug category, priority, and version to

determine whether the bug report is duplicate or not using the textual, categorical, and contextual similarity metrics.

3.2. ML-based Approaches

In [22], Rodrigues et al. proposed a novel approach called *Soft Alignment Method for Bug Deduplication* (SABD). Their approach is based on deep learning and uses two sub-neural networks. One network deals with categorical fields in bug reports (version, component name, etc.), and the other deals with free textual information. Their approach receives two bug reports: query report (new report) and a candidate one (previously submitted report). The outputs of the two sub-networks are concatenated to determine whether the candidate bug report is duplicated or not. In [23], Deshmukh et al. proposed a deep learning architecture consisting of CNN and Siamese LSTM to detect duplicate bug reports. Three different types of encoders are used to encode different bug report features. The outputs are concatenated to represent the bug report. In [24], He et al. used a dual-channel convolutional neural network (DC-CNN) with the word2vec (semantic weighting mechanism) to detect duplicate bug reports. In [25], Haering et al. combined BERT and deep learning for bug report detection.

In [26], Sun et al. used the support vector machine (SVM) to detect duplicate bug reports. In their work, a discriminative model is built to check whether two bug reports are duplicated or not based on probability scores. In [27], Klein et al. investigated different machine-learning algorithms to classify duplicate bug reports. These classifiers are Random Forest, KNN, Decision Tree, Linear SVM, Naive Bayes, and RBF.

All the above-mentioned approaches in this section are designed to detect only duplicate issues and ignore detecting similar issues that are not duplicated. The proposal presented in this article is to detect and cluster similar issues together.

4. The Proposed Approach

The clustering process in this research work is guided by two important features extracted from the issue's report. These features are the *title* and *body* of each issue. When a contributor submits an issue, the issue tracking system asks the contributor to provide *title*, *body*, and *label* of the submitted issue. These features serve as clues to link similar issues together. The issue's title and label is a brief text and includes valuable, and precise words about the issue. The issue's body is descriptive text to detail the issue with technical steps and code snippets (see Figure 2). In this study, title and body features are only considered during the clustering process while Label is ignored as it is repetitive and missed in many issues.

Figure 3 displays a holistic view of the proposed approach to cluster similar issues together. For a given GitHub's repository, the issues with all their contents (title, body, label, etc.) are gathered using GraphQL API. This content is parsed and only the features of interest are selected for each issue individually. Then, The text of the extracted features are pre-processed. Next, each issue is vectorized and weighted. Finally, the AHC algorithm is applied to group similar issues together. The obtained clusters are compared to reference clusters to evaluate the effectiveness of the proposed approach.

4.1. Parsing and Feature Selection

Feature selection is a process to remove unneeded features that could increase the computation cost and degrade the results of the clustering model. Therefore, in this step, the textual content of each individual issue is parsed to keep only the important features and discard the rest. As mentioned above, these important features are the title and body of each issue.

4.2. Data Preprocessing

In this step, standard pre-processing tasks in natural language processing (NLP) are followed. These include removing special characters and punctuation marks, splitting the issue's text into tokens with lower casing, removing stop words, and stemming.

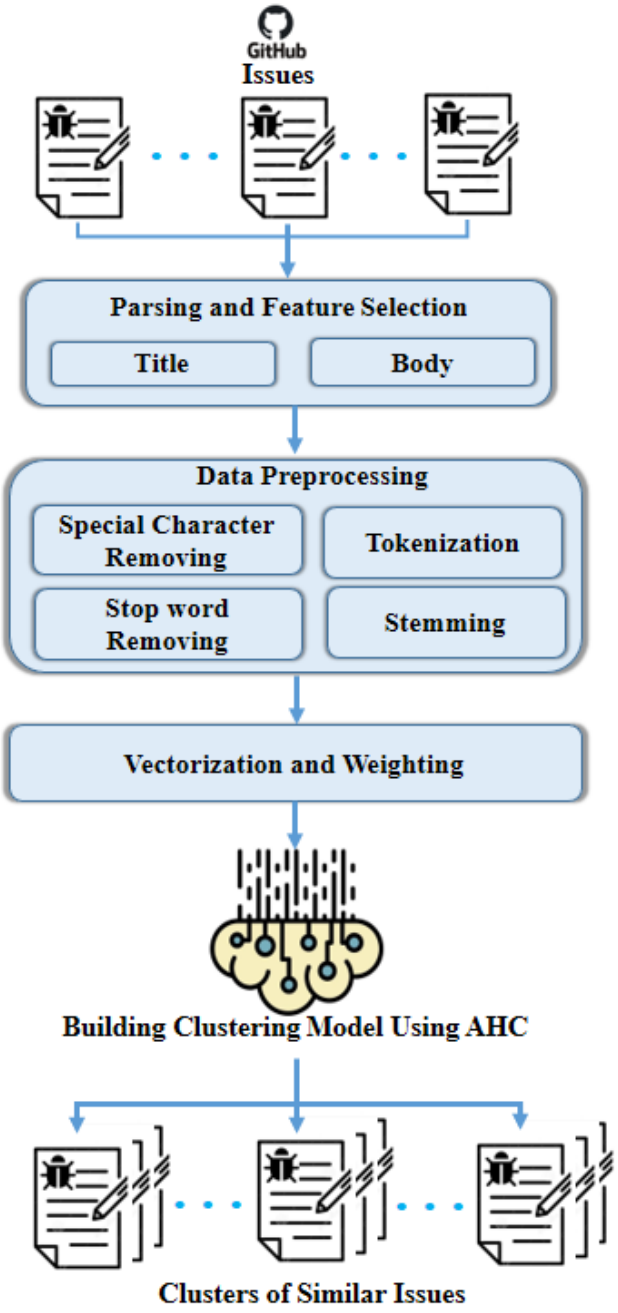

Figure 3: Clustering process overview.

4.2.1. Special Characters Removing

This task aims to remove numbers, symbols, punctuation marks, parentheses, and HTTP links from the title and body of each issue. To complete this cleaning task, an appropriate regular expression has been applied using "re" library in Python.

4.2.2. Tokenization and Lower Casing

This task aims to split the title and body text of each issue into separate words called tokens. This splitting task is based on space (as a separator) and camel case notation. This task is completed using the "word_tokenize" function from the NLTK (the Natural Language Toolkit) library in Python. Then, each identified token is lowered case.

4.2.3. Stop Word Removing

After the tokenization task has been completed, not all resulting tokens are viable. Therefore, in this step, stop words (of, the, on, while, etc.) are removed. These words appear frequently in the title and body of each issue and are not important to find textual similarity among issues. To achieve this step, the NLTK library in Python is used.

4.2.4. Stemming

Tokens extracted from all issues are the base to compute textual similarity among issues' vectors. Therefore, tokens must be returned back to their base forms (stems) in a process called stemming. In this step, the most widely used stemming algorithm in English is used and called "Porter stemmer". This algorithm is implemented by NLTK library in Python.

4.3. Vectorization and Weighting

To easily compare issues, a vector with tokens is created for each issue in a process called vectorization. Each token in the issue's vector has a weight to represent the importance of that token in that vector (issue). In this study, a well-known term weighting metric in NLP is used called TF-IDF (Term Frequency Inverse Document Frequency) [28]. In this metric, a global weighting score is assigned to each token in the issue corpus, which consists of all tokens extracted from all feature texts for all issues. This score takes into account not only the frequency of a token in individual issues but also across all issues. The weight score rises when a token appears more frequently in a certain issue, but it decreases when the token appears more frequently in different issues. Additionally, this score reveals the significance of a token in a particular issue; higher TF-IDF scoring tokens are better in expressing the issue's content. To complete this step, a TF-IDF matrix is built. In this matrix, all tokens in the issues' corpus are rows, all issues' vectors are columns, and the cell at the intersection of a selected token and the issue's vector indicates to TF-IDF weight of that token in that vector. The TF-IDF weight score is computed using the equations 1 - 3 and this victorization process is achieved using the "TfidfVectorizer" function implemented in the "scikit-learn" library.

$$TFIDF(t, d) = \text{tf(t, d)} * log\frac{N}{(df + 1)} \tag{1}$$

$$tf(t, d) = \frac{\text{count of t in d}}{\text{number of words in d}} \tag{2}$$

$$df(t) = \text{occurrence of t in N document} \tag{3}$$

Where N is the number of issues' vectors in the corpus. *tf* and *df* are term frequency and document frequency (vector frequency), respectively. t and d are token and document (vector), respectively.

4.4. Building Clustering Model using AHC

Any clustering algorithm relies on some measure of similarity to group similar data points in the space together. In this study, cosine similarity is adopted as a similarity measure to build a clustering model using AHC. It is a widely used metric over others working on textual artifacts in different software engineering tasks. In this research work, it is used to compute the textual similarity between any two issues' vectors in the TF-IDF matrix (built in the previous step). It works by computing the cosine of the angle between these vectors. Its score takes a range between [0, 1] as it is computed only over the positive space. According to this measure, *0* refers to weak similarity, while *1* refers to strong similarity between issues' vectors. These cosine scores are computed using the equation 4.

$$Cosine(V_1, V_2) = cos(\theta) = \frac{\overrightarrow{V_1}.\overrightarrow{V_2}}{\|\overrightarrow{V_1}\|\|\overrightarrow{V_2}\|} \tag{4}$$

The next subsections explain the main two steps in the AHC algorithm to build a clustering model that groups similar issues' vectors into clusters based on their cosine similarity scores.

4.4.1. Clustering and its Visualization

The general application of AHC algorithm requires building the *dendrogram* tree of initial singleton clusters. In this research work, a dendrogram tree of TF-IDF vectors is built (issues' vectors) according to the Algorithm 1. The input of this algorithm is all issues' vectors in TF-IDF matrix, and the output is a nested dendrogram tree of these vectors. The algorithm starts with a set of singleton clusters, where initially each issue's vector is a singleton cluster. Then, the most similar vectors are merged together to form a new cluster replacing them (in each round of the while-loop (lines 2–7). This nested binary merging proceeds through while-loop rounds to finally get a single cluster including all issues' vectors.

Figure 4 demonstrates a simple example to build a dendrogram tree of a set of issue vectors (15 issue vectors). In this figure, the number of these vectors is placed on the x-axis, while distance scores (cluster distance) are placed on the y-axis. As shown in this tree structure, the initial level (seed or bottom level) consists of 15 single clusters. When you go up in the tree, each closest pair of child issues is merged together to form a parent of these children. This binary merging process continues until a single cluster (parent) consisting of all issues is obtained.

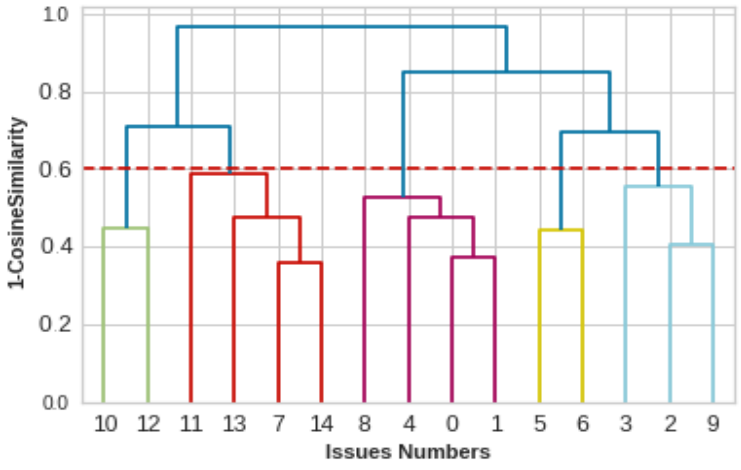

Figure 4: An example of dendrogram tree with cutting line.

Algorithm 1: General Algorithm for Building Dendrogram Tree

```
   Input: ISs                //IS Vectors
   Output: dendgro           //Dendrogram Tree
 1 stack seeds ← ISs
 2 while ( |seeds| > 1) do
 3 |   (Clu1,Clu2) ← mostSimilarClusters(seeds)
 4 |   Pop(Clu1, seeds)      //Clu1: Cluster 1
 5 |   Pop(Clu2, seeds)      //Clu2: Cluster 2
 6 |   Clu3← Combine(Clu1, Clu2)
 7 |   Push(Clu3, seeds)
 8 end
 9 dendgro ← seeds
10 return dendgro
```

4.4.2. Extracting Clusters of Similar Issues

In the AHC algorithm, the dendrogram tree is built without a prior assumption about the number of clusters. Therefore, the dendrogram tree should be horizontally cut to produce a set of interested clusters. The horizontal cutting point represents the cluster's distance threshold specified by the repository owner. When this cutting point goes down or up in the dendrogram, the number of clusters increases or decreases, respectively. Each child branch formed below the horizontal cutting line represents a disjoint individual cluster. Therefore, this cutting point should be specified by the repository owner who is aware and can evaluate which issues are similar. In this research work, this cutting point is set after many tries to find reasonable similar issue clusters based on title and description matching.

Algorithm 2 is proposed to extract potential clusters of similar issues based on the depth-first search (DFS) algorithm. The algorithm has two inputs: the constructed dendrogram tree (dend), and the cluster distance threshold (DisT). The essence of this algorithm is to compare the cosine similarity of right and left child clusters starting from the parent root of the dendrogram, where each child cluster is considered a single vector. If the computed similarity score is less than the distance threshold (DisT), the search goes down in the dendrogram tree to the next immediate branches. Otherwise, the parent cluster is added to an accumulator of potential clusters (called IsClusters) as a potential cluster of similar issues. This searching process in the dendrogram tree continues until all potential clusters of similar issues are identified.

By referring again to the dendrogram tree presented in Figure 4, the red horizontal line in this tree represents the cutting point which is defined based on a distance threshold score equal to (60%). Based on this cut point, there are five potential clusters of similar issues identified: [10,12], [11,13,7,14], [8,4,0,1], [5,6], and [3,2,9]. The issues/nested clusters with the same height belong to the same clusters.

5. Experimental Results and Evaluation

In this section, the effectiveness of the proposed approach is evaluated using a ground-truth dataset. In the beginning, the dataset used for evaluation is described. Then, research questions and evaluation procedures are listed. After that, the experimental results are presented with discussion. Finally, the threats to validity are refuted.

5.1. Dataset

To evaluate the effectiveness of the proposed approach, it should be applied to a ground-truth dataset in the subject. However, such a dataset does not exist yet. Therefore, the proposed approach is evaluated using a dataset of duplicate clusters of issues, as 100% duplicate issues are similar issues, too.

The proposed approach is applied to a public dataset of duplicate issues from different GitHub repositories and software projects called BugRepo (freely available on https://github.com/logpai/bugrepo). Randomly,

Algorithm 2: Identifying Similar Clusters of Issues

```
Input: dend, DisT          // a dendrogram tree of FRs and distance threshold
Output: IsClusters         // a set of issues clusters
stack heap                 // heap is a stack
heap.push(root(dend))
while (|heap|> 0) do
    parent ← heap.pop()
    Clu1 ← getLeftCluster(parent, dend)
    Clu2 ← getRightCluster(parent, dend)
    Sim ← cosSim(Clu1, Clu2)
    if (Sim > DisT) then
        IsClusters.add(Clu1)
        IsClusters.add(Clu2)
    else
        heap.push(Clu1)
        heap.push(Clu2)
    end
end
return IsClusters
```

different numbers and sizes of duplicate clusters with two or more issues are chosen from different software components (as shown in Table 1). These chosen clusters are used as ground-truth clusters to evaluate the proposed approach in this study. As shown in this table, these clusters are diverse to ensure free-bias evaluation.

The ground-truth clusters used in the evaluation are described statistically in Table 1. It shows, for each project/software component, the total number of duplicate issues (#ISs), as well as the number and size of ground-truth duplicate clusters of issues (#GTC). This table demonstrates that there are *13* projects with *399* issues and *99* clusters that are taken into account during the evaluation process. These clusters represent ground-truth clusters in the evaluation process.

During this section, box-plots are used to better understand the obtained results. This type of plotting visualizes the results as boxes to show their distribution and centrality. The box consists of different statistical measures: min, lower quartile (Q1), median, upper quartile (Q3), and max.

5.2. Research Questions and Evaluation Procedures

In this subsection, the research questions to answer are listed with the evaluation metrics.

5.2.1. ***RQ1: To what extent are the identified issue clusters correct?***

This research question aims to measure the correctness of each identified cluster of issues. This means to what extent the member issues of a cluster are related to each other (similar). To address this research question, a measure from information retrieval (IR) is adapted to answer it. This measure is called Precision (P) and is adapted as follows:

$$Precision = \frac{|EXC \cap GTC|}{|EXC|} \times 100\% \tag{5}$$

Where EXC and GTC are **ex**tracted **c**luster of issues and **g**round-**t**ruth **c**luster of issues, respectively. GTCs are obtained by referring to a ground-truth dataset in the subject (see the dataset subsection). Precision values take a range of [0-1]. The higher Precision value you have, the higher correctness cluster is identified, and vice-versa. The procedure below is followed to answer this research question:

Table 1: Statistical information of ground-truth clusters with their duplicate issues.

Repository Name	#ISs	#Clusters (GTC)	Cluster Size		
			Min	Max	Avg
JTD-Core	34	7	3	8	4.85
JTD-UI	57	12	2	7	4.72
JTD-Debug	55	11	3	7	5
JTD-Doc	8	3	2	4	2.66
JTD-Text	20	9	2	4	2.66
EclipsePlatfrom-SWT	26	6	3	7	4.33
EclipsePlatfrom-UI	41	7	3	10	5.86
EclipsePlatfrom-Debug	26	7	2	5	3.71
EclipsePlatfrom-Doc	20	8	2	5	2.5
Thrunderbird-Security	10	4	2	3	3.5
Thrunderbird-General	32	10	2	5	3.2
Thrunderbird-Installer	23	7	2	6	3.28
MozillaFirefox-Search	47	8	2	12	5.88

1- For each extracted cluster (EXC), its corresponding ground-truth cluster is picked from the dataset. A cluster from the dataset that shares a maximum number of issues with the extracted cluster is the target and is called a ground-truth cluster.

2- Then, the Precision equation is applied to compute the Precision value of that extracted cluster.

5.2.2. ***RQ2: To what extent are the identified issue clusters complete?***

In this research, the goal is to investigate to what extent the member issues of each identified cluster are complete (i.e., there are no missing relevant member issues in a given EXC). To address this research question, a completeness measure from IR is adapted and used. This measure is called and is adapted as follows:

$$Recall = \frac{|EXC \cap GTC|}{|GTC|} \times 100\% \tag{6}$$

Where EXCs and GTC are **ex**tracted **c**luster of issues and **g**round-**t**ruth **c**luster of issues, respectively. Like Precision, GTCs are obtained by referring to a ground-truth dataset in the subject (see the dataset subsection). Recall values take a range of [0-1]. The higher Recall value you have, the higher completeness cluster is extracted, and vice-versa. The procedure below is followed to answer this research question:

1- For each extracted cluster (EXC), its corresponding ground-truth cluster is picked from the dataset. A cluster from the dataset that shares a maximum number of issues with the extracted cluster is the target and is called a ground-truth cluster.

2- Then, the Recall equation is applied to compute the Recall value of that identified cluster.

Table 2: Hyperparameters of AHC algorithm.

Hyperparameter	**AHC**	**TfidfVectorizer**
Stop-words	-	English
Max-Features	None	None
Affinity	euclidean	-
Linkage	ward	-
Distance_Threshold	Calibrated	-

5.2.3. ***RQ3: To what extent are the identified issue clusters accurate?***

This research question is designed to know to what extent the extracted cluster includes all relevant issues and only those issues in the same cluster without interfering with issues from other clusters. To address this research question, F-measure measure from IR field is used to compute the accuracy value for each extracted cluster using the following formula:

$$F - measure = \frac{2.Precision.Recall}{Precision + Recall} \times 100\% \tag{7}$$

F-measure makes a trade-off between Precision and Recall. The higher Precision and Recall value you have, the higher F-measure value you will obtain, and vice-versa. The F-measure takes a range of [0 - 1].

5.2.4. ***RQ4: How much the reviewing efforts could be saved?***

This research question focuses on the reviewing effort saved when similar Issues are grouped together and assigned to a single relevant reviewer or team. To address this research question, a measure from a previous work [10] is used to quantify the saved effort of the reviewer(s) (see equation 8). According to this measure, if EXC consists of 10 issues, then SRE (saved reviewing effort) is 90% (1-(1/10)). However, if these issues are distributed individually to different reviewers (for example 10 reviewers), the SRE value will be reduced. The SRE value takes a range of [0–1]. The higher the SRE value you get, the higher saved effort results.

$$SRE = 1 - \frac{1}{|EXC|} \times 100\% \tag{8}$$

5.3. Results

To get answers to the proposed research questions, the AHC algorithm is applied to each repository/software component in the dataset. AHC is implemented by *"AgglomerativeClustering"* method from the "scikit-learn" library in Python. Also, TF-IDF matrix is built using *"TfidfVectorizer"* provided by the "scikit-learn" library. The hyperparameters of *"AgglomerativeClustering" and "TfidfVectorizer"* methods are shown in Table 2. In this table, the English dictionary of the "scikit-learn" library is used to remove stop words. The number of features used to build TF-IDF matrix is automatically selected by "TfidfVectorizer" method. The affinity parameter is a metric to compute the distance among issue vectors and its value is "euclidean". The linkage parameter is to determine which pair of clusters should be merged and its value is "ward". The distance threshold determines a threshold above which, the clusters will not be merged. This threshold is calibrated by the repository owner. Then, average Precision, Recall, and F-measure values of all EXCs from each repository are computed and placed in Tables 3, 4, and 5, respectively.

5.3.1. ***RQ1: To what extent are the identified issue clusters correct?***

Table 3 shows Precision values of identified clusters of issues using the AHC algorithm based on selected features (title and description). As shown in this table, Precision values on average take a good range [0.67–0.92], with a maximum value always equal to 1.0. Obviously, according to the results of this table, the threshold distance can always be celebrated to get the number of EXCs equal to the its corresponding number of GTCs regardless of the number of issues.

Table 3: Precision values of the extracted clusters using AHC.

Repository Name	#ISs	#GTCs	#EXCs	Precision			
				Min	**Max**	**Avg**	**StdDev**
JTD-Core	34	7	7	0.40	1.0	0.67	0.25
JTD-UI	57	12	12	0.60	1.0	0.92	0.13
JTD-Debug	55	11	11	0.38	1.0	0.79	0.21
JTD-Doc	8	3	3	0.6	1.0	0.87	0.18
JTD-Text	20	9	9	0.5	1.0	0.82	0.20
EclipsePlatform-SWT	26	6	6	0.56	1.0	0.88	0.18
EclipsePlatform-UI	41	7	7	0.45	1.0	0.92	0.19
EclipsePlatform-Debug	26	7	8	0.50	1.0	0.75	0.17
EclipsePlatform-Doc	20	8	8	0.50	1.0	0.90	0.18
Thrunderbird-Security	10	4	4	0.67	1.0	0.92	0.14
Thrunderbird-General	32	10	10	0.43	1.0	0.83	0.22
Thrunderbird-Installer	23	7	7	0.42	1.0	0.83	0.21
MozillaFirefox-Search	47	8	8	0.40	1.0	0.87	0.23

Figure 5 shows box-plots of Precision values for all identified clusters from each considered repository. The Y-axis shows Precision values, the X-axis shows repository names, and for each repository, there is its own box-plot. It is clear from this figure that (75%) of the identified clusters from each repository (except JTD-Core) have Precision values of more than 0.62 up to 1.0. Also, 25% of the identified clusters across all repositories are either outliers or have a minimum value far from the box-plot. Moreover, there are different repositories with short box-plots and minimum values are merged into the box-plots. This means that the Precision values of identified clusters from these repositories are similar (Q1≈ Q2≈ median≈ Q3≈Q4).

5.3.2. ***RQ2: To what extent are the identified issue clusters complete?***

Table 4 demonstrates the average Recall results of issue clusters extracted from each repository. As seen in the table, Recall values typically fall within the range [0.65 - 0.92], with always 1.0 as the maximum value. Also, the results of this table clearly show that, regardless of the number of available issues, the threshold distance can always be adjusted to obtain an equal number of EXCs and GTCs. Moreover, minimum Precision values in some repositories are low, but these values are outliers, as we will see in box-plots next.

Figure 6 shows box-plots of Recall values for all identified clusters from each separate repository. The Y-axis displays Recall values, the X-axis displays repository names, and there is a separate box-plot for each repository. In the most of considered projects, 75% of the Recall values of the extracted clusters are in a range [70% - 100%]. Also, the minimum Recall values of extracted clusters across all projects are far from the boxes or outliers.

5.3.3. ***RQ3: To what extent are the identified issue clusters accurate?***

Table 5 shows the results of F-measure for extracted issue clusters across all considered projects. The results presented in this table demonstrate that average F-measure values fall in a high range [60% - 91%] with a maximum value of 100%. Moreover, the minimum values take a reasonable range [40% - 80%]. As seen in this table, the number of extracted clusters in each project is equal to the number of GTCs in that

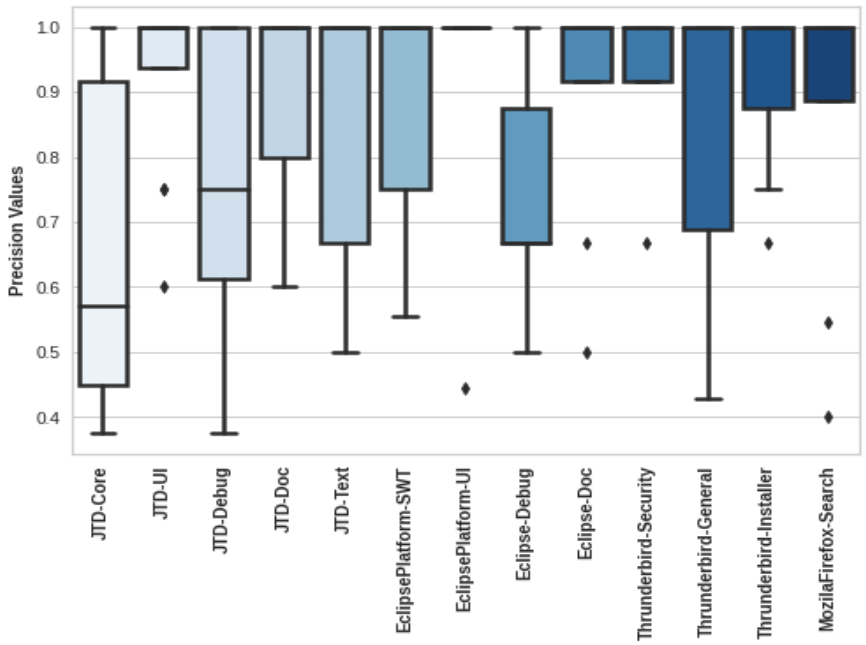

Figure 5: Distribution of Precision values against each software project component.

Table 4: Recall values of the extracted clusters using AHC.

Repository Name	#ISs	#GTCs	#EXCs	**Recall**			
				Min	**Max**	**Avg**	**StdDev**
JTD-Core	34	7	7	0.34	1.0	0.67	0.23
JTD-UI	57	12	12	0.60	1.0	0.91	0.13
JTD-Debug	55	11	11	0.5	1.0	0.74	0.15
JTD-Doc	8	3	3	0.25	1.0	0.67	0.31
JTD-Text	20	9	9	0.33	1.0	0.65	0.21
EclipsePlatform-SWT	26	6	6	0.50	1.0	0.80	0.17
EclipsePlatform-UI	41	7	7	0.40	1.0	0.80	0.21
EclipsePlatform-Debug	26	7	8	0.40	1.0	0.67	0.18
EclipsePlatform-Doc	20	8	8	0.50	1.0	0.88	0.21
Thrunderbird-Security	10	4	4	0.67	1.0	0.92	0.14
Thrunderbird-General	32	10	10	0.33	1.0	0.80	0.21
Thrunderbird-Installer	23	7	7	0.34	1.0	0.80	0.22
MozillaFirefox-Search	47	8	8	0.58	1.0	0.86	0.15

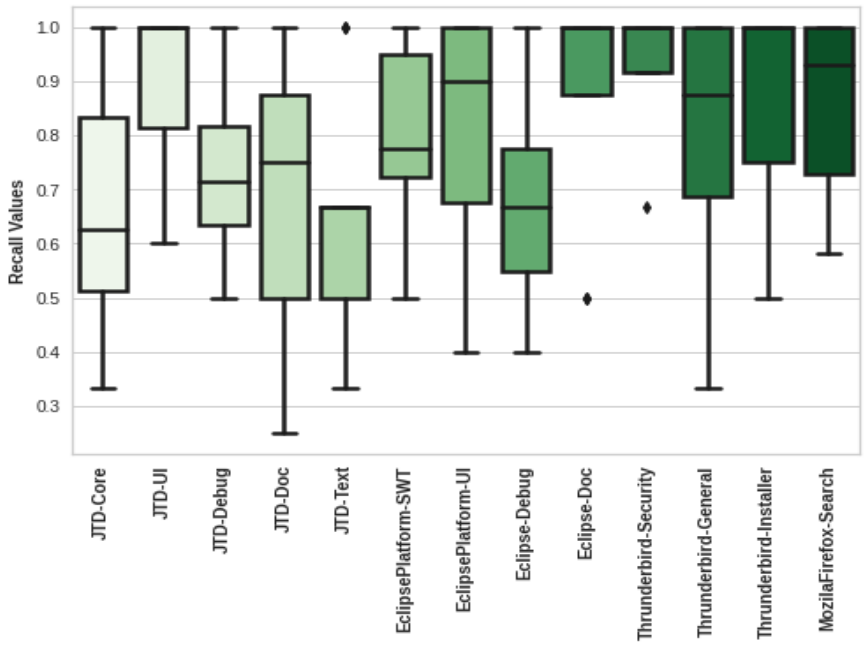

Figure 6: Distribution of Recall values against each software project component.

project. Figure 7 shows box-plots corresponding to F-measures values of extracted clusters in each project. From the shown boxes, 75% of F-measure values of extracted clusters, in the majority of projects, have a high range [70% - 100%] while minimum values are far from the boxes or outliers.

5.3.4. ***RQ4: How much the reviewing efforts could be saved?***

Table 6 displays the results of reviewing efforts that could be saved by using the proposed approach. In the last column in this table (SRE), these saved efforts values are presented. These values are computed for extracted issue clusters that have the maximum number of issues in each considered project. It can be seen in this table that the SRE value has a high range of [66.6% - 99.9%].

5.4. Discussion

The obtained experimental results, as shown in Figures 5, 6 and 7, firstly indicate that 75% of the Precision values of extracted clusters from each project fall in a range [61% - 100%] with median = Q3 = max in the majority of them. However, Precision values of extracted clusters only from the "JTD-Core" component take a range of [45% - 91%]. Secondly, 75% Recall values of the extracted clusters from the majority of projects have a range of [60% - 100%] while, for other projects, such as "JTD-Text" and JTD-Core, these values have a range of [50% - 83%]. Thirdly, F-measure results confirm that there is a minor degradation in the extraction of the duplicate clusters from "JTD-Core" and "JTD-Text" as their 75% F-measure values have a range [53% - 72%] and [67% - 80%] with maximum values near to the box, respectively. This minor degradation in the results of extracted clusters from "JTD-Core" and "JTD-Text" is due to the following reasons: different vocabulary used and short issue titles.

- **Different Vocabulary Used.** After a manual investigation of the description of "JTD-Core" and "JTD-Text" issues, sometimes issue submitters use different vocabularies to define the submitted issues, especially in the description field of the issue. This case normally occurs because the submitters have different technical backgrounds and cultures.

- **Short Issues Titles.** Also, after manual investigation of issue titles of both "JTD-Core" and "JTD-Text", some duplicate issues have short titles after removing stop words (i.e., titles with only two or

Table 5: F-measure values of the extracted clusters using AHC.

Repository Name	#ISs	#GTCs	#EXCs	**F-measure**			
				Min	**Max**	**Avg**	**StdDev**
JTD-Core	34	7	7	0.40	0.75	0.60	0.12
JTD-UI	57	12	12	0.67	1.0	0.91	0.12
JTD-Debug	55	11	11	0.55	1.0	0.75	0.14
JTD-Doc	8	3	3	0.40	1.0	0.69	0.24
JTD-Text	20	9	9	0.40	1.0	0.70	0.16
EclipsePlatform-SWT	26	6	6	0.63	1.0	0.81	0.15
EclipsePlatform-UI	41	7	7	0.57	1.0	0.82	0.16
EclipsePlatform-Debug	26	7	8	0.50	1.0	0.70	0.16
EclipsePlatform-Doc	20	8	8	0.50	1.0	0.87	0.18
Thrunderbird-Security	10	4	4	0.80	1.0	0.90	0.09
Thrunderbird-General	32	10	10	0.40	1.0	0.80	0.19
Thrunderbird-Installer	23	7	7	0.67	1.0	0.86	0.13
MozillaFirefox-Search	47	8	8	0.50	1.0	0.84	0.16

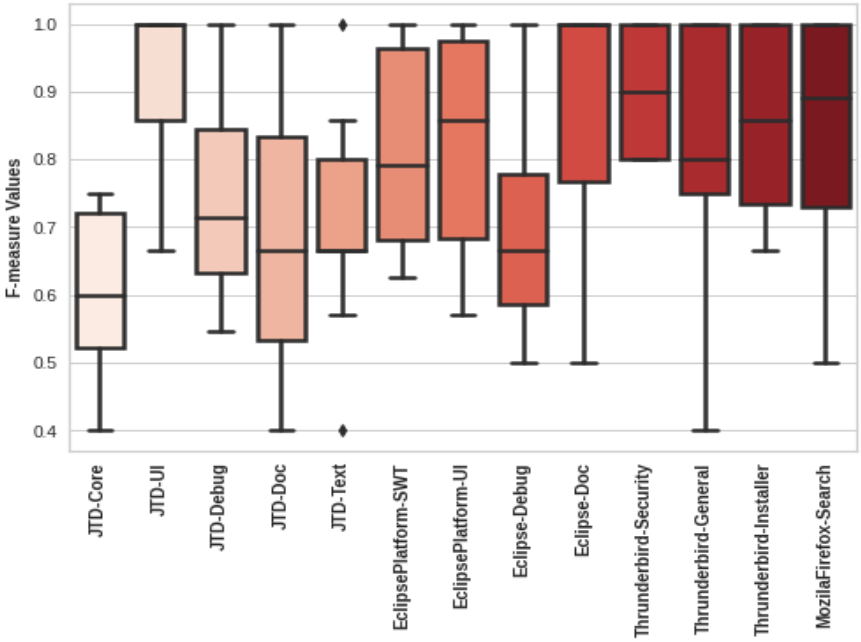

Figure 7: Distribution of F-measure values against each software project component.

Table 6: Values of saving reviewing efforts across all considered software projects.

Repository Name	#ISs	#GTCs	#EXCs	Cluster Size			SRE
				Min	Max	Avg	
JTD-Core	34	7	7	2	8	4.86	87.5%
JTD-UI	57	12	12	2	8	4.75	87.5%
JTD-Debug	55	11	11	2	8	5	87.5%
JTD-Doc	8	3	3	1	5	2.7	80%
JTD-Text	20	9	9	1	4	2.22	75%
EclipsePlatform-SWT	26	6	6	2	9	4.33	88.8%
EclipsePlatform-UI	41	7	7	3	9	5.86	88.8%
EclipsePlatform-Debug	26	7	8	3	6	3.71	83.3%
EclipsePlatform-Doc	20	8	8	1	5	2.5	80%
Thrunderbird-Security	10	4	4	2	3	2.5	66.6%
Thrunderbird-General	32	10	10	2	7	3.2	85.7%
Thrunderbird-Installer	23	7	7	3	4	3.28	75%
MozillaFirefox-Search	47	8	8	2	11	5.88	90.9%

three words), especially in"JTD-Text" component. Whenever the input of the proposed is insufficient, this leads to misleading the clustering algorithm used.

Obviously, the results displayed in Table 6 show that the amount of reviewing effort for issues that are saved in a given repository depends on the size of the extracted issue clusters from that repository. When the size of the correct extracted clusters (in terms of evaluation metrics) increases, SRE values rise, too.

As a summary, the AHC algorithm can extract relevant similar issue clusters from GitHub repositories depending on only two issue features: title and description. Also, the distance threshold used by AHC can be easily calibrated to return a number of relevant issue clusters equal to the number of ground-truth clusters. Moreover, Precision, Recall and F-measure values of extracted clusters are relatively high in the majority of considered projects. This is depending on the results of Table 3, 4, and 5.

5.5. Threats to Validity

As with any research work, this study has threats to validity. These threats are as follows:

- **Internal threat.** The internal threat refers to issues related to the computational steps of the proposed approach and thus the results credability. In this study, when the issues submitter uses different vocabularies in the issue text (title and description), the values of evaluation metrics may degrade. This is because the textual similarity among similar issues is missed, and thus they are scattered across different clusters. However, this limitation is a common concern among all textual similarity-based approaches.

- **External threat.** The external threat refers to concerns that hinder the obtained results generalization. In this study, the proposed approach is targeted to find and cluster similar issues into clusters in each individual repository. However, there are similar issues across multiple repositories in the same domain (e.g., Android applications) which are overlooked by the proposed approach. In fact, the approach can be applied across repositories as it is a textual similarity-based approach and not an application-based approach.

6. Conclusion and Future Work

In this paper, a machine learning-based approach is proposed to support the issue tracking system on GitHub via grouping similar submitted issues together. The proposed approach helps to speed up the assigning process of similar issues, and thus speed up fixing them, whereas assigning them individually to different developers introduces inconsistencies in addition to slowing issues fixing. The approach takes, as input, issues of a given repository. Then, it computes the textual similarity among these issues. Issues' titles and descriptions are exploited to find this similarity. This textual information represents features to help machine learning algorithms to cluster similar issues together. Numerous software components from different repositories have been used to evaluate the approach. Findings show that the proposed approach identifies similar clusters of issues with promising results using widely used evaluation measures in this subject: Precision, Recall, and F-measure.

In future work, the results of using other issues' features will be investigated using semantic similarity measures in natural language processing such as BERT (Bidirectional Encoder Representations from Transformers) [29].

References

[1] R. Kallis, A. Di Sorbo, G. Canfora, S. Panichella, Ticket tagger: Machine learning driven issue classification, in: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 406–409. doi:10.1109/ICSME.2019.00070.

[2] User search, GitHub, online, accessed 01/11/2022, search result shows more than 100M users (November 2022).

[3] Github Number of Repository Search, https://github.com/search, online, accessed 01/11/2022, search result shows more than 352M repositories (November 2022).

[4] Q. Fan, Y. Yu, G. Yin, T. Wang, H. Wang, Where is the road for issue reports classification based on text mining?, in: Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '17, IEEE Press, 2017, p. 121–130. doi:10.1109/ESEM.2017.19.
URL https://doi.org/10.1109/ESEM.2017.19

[5] M. Izadi, K. Akbari, A. Heydarnoori, Predicting the objective and priority of issue reports in a cross project context, CoRR abs/2012.10951. arXiv:2012.10951.
URL https://arxiv.org/abs/2012.10951

[6] M. Borg, L. Jonsson, E. Engstrom, B. Bartalos, A. Szab'o, Adopting automated bug assignment in practice: A longitudinal case study at ericsson, ArXiv abs/2209.08955.
URL https://api.semanticscholar.org/CorpusID:252367439

[7] Z. Liao, D. He, Z. Chen, X. Fan, Y. Zhang, S. Liu, Exploring the characteristics of issue-related behaviors in github using visualization techniques, IEEE Access 6 (2018) 24003–24015, acceptance from VoR OA article however no CC licence on article (see p1 of VoR). Applied 'no exception' as article doesn't meet our definition for Gold exception. ET 14/1/20 ET. doi:10.1109/ACCESS.2018.2810295.

[8] J. Cabot, J. L. C. Izquierdo, V. Cosentino, B. Rolandi, Exploring the use of labels to categorize issues in Open-Source Software projects, in: Y.-G. Guéhéneuc, B. Adams, A. Serebrenik (Eds.), Proceedings of the 22nd International Conference on Software Analysis, Evolution and Reengineering, IEEE, 2015, pp. 550–554. doi:10.1109/SANER.2015.7081875.

[9] K. Herzig, S. Just, A. Zeller, It's not a bug, it's a feature: How misclassification impacts bug prediction, in: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, 2013, p. 392–401.

[10] H. Eyal Salman, Z. Alshara, A.-D. Seriai, Automatic identification of similar pull-requests in github's repositories using machine learning, Information 13 (2). doi:10.3390/info13020073.
URL https://www.mdpi.com/2078-2489/13/2/73

[11] A. E. Ezugwu, A. M. Ikotun, O. O. Oyelade, L. Abualigah, J. O. Agushaka, C. I. Eke, A. A. Akinyelu, A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects, Engineering Applications of Artificial Intelligence 110 (2022) 104743. doi:https://doi.org/10.1016/j.engappai.2022.104743.
URL https://www.sciencedirect.com/science/article/pii/S095219762200046X

[12] H. Zhao, Z. Qi, Hierarchical agglomerative clustering with ordering constraints, in: 2010 Third International Conference on Knowledge Discovery and Data Mining, 2010, pp. 195–199. doi:10.1109/WKDD.2010.123.

[13] S. Gupta, S. K. Gupta, A systematic study of duplicate bug report detection, International Journal of Advanced Computer Science and Applications 12 (1). doi:10.14569/IJACSA.2021.0120167.
URL http://dx.doi.org/10.14569/IJACSA.2021.0120167

[14] I. Chawla, S. K. Singh, Performance evaluation of vsm and lsi models to determine bug reports similarity, in: 2013 Sixth International Conference on Contemporary Computing (IC3), 2013, pp. 375–380. doi:10.1109/IC3.2013.6612223.

[15] P. Runeson, M. Alexandersson, O. Nyholm, Detection of duplicate defect reports using natural language processing, in: 29th International Conference on Software Engineering (ICSE'07), 2007, pp. 499–510. doi:10.1109/ICSE.2007.32.

[16] X. Wang, L. Zhang, T. Xie, J. Anvik, J. Sun, An approach to detecting duplicate bug reports using natural language and execution information, in: Proceedings of the 30th International Conference on Software Engineering, ICSE '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 461–470. doi:10.1145/1368088.1368151.
URL https://doi.org/10.1145/1368088.1368151

[17] M. Amoui, N. Kaushik, A. Al-Dabbagh, L. Tahvildari, S. Li, W. Liu, Search-based duplicate defect detection: An industrial experience, in: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, IEEE Press, 2013, p. 173–182.

[18] F. Thung, P. S. Kochhar, D. Lo, Dupfinder: Integrated tool support for duplicate bug report detection, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Association for Computing Machinery, New York, NY, USA, 2014, p. 871–874. doi:10.1145/2642937.2648627.
URL https://doi.org/10.1145/2642937.2648627

[19] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, C. Sun, Duplicate bug report detection with a combination of information retrieval and topic modeling, in: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, Association for Computing Machinery, New York, NY, USA, 2012, p. 70–79. doi:10.1145/2351676.2351687.
URL https://doi.org/10.1145/2351676.2351687

[20] T. Akilan, D. Shah, N. Patel, R. Mehta, Fast detection of duplicate bug reports using lda-based topic modeling and classification, in: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2020, pp. 1622–1629. doi:10.1109/SMC42975.2020.9283289.

[21] A. Hindle, A. Alipour, E. Stroulia, A contextual approach towards more accurate duplicate bug report detection and ranking, Empirical Softw. Engg. 21 (2) (2016) 368–410. doi:10.1007/s10664-015-9387-3.
URL https://doi.org/10.1007/s10664-015-9387-3

[22] I. M. Rodrigues, D. Aloise, E. R. Fernandes, M. Dagenais, A soft alignment model for bug deduplication, in: Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 43–53. doi:10.1145/3379597.3387470.
URL https://doi.org/10.1145/3379597.3387470

[23] C. Sun, D. Lo, S.-C. Khoo, J. Jiang, Towards more accurate retrieval of duplicate bug reports, in: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011, pp. 253–262. doi:10.1109/ASE.2011.6100061.

[24] J. He, L. Xu, M. Yan, X. Xia, Y. Lei, Duplicate bug report detection using dual-channel convolutional neural networks, in: Proceedings of the 28th International Conference on Program Comprehension, ICPC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 117–127. doi:10.1145/3387904.3389263.
URL https://doi.org/10.1145/3387904.3389263

[25] M. Häring, C. Stanik, W. Maalej, Automatically matching bug reports with related app reviews (2021). doi:10.48550/ARXIV.2102.07134.
URL https://arxiv.org/abs/2102.07134

[26] C. Sun, D. Lo, X. Wang, J. Jiang, S.-C. Khoo, A discriminative model approach for accurate duplicate bug report retrieval, in: 2010 ACM/IEEE 32nd International Conference on Software Engineering, Vol. 1, 2010, pp. 45–54. doi:10.1145/1806799.1806811.

[27] N. Klein, C. S. Corley, N. A. Kraft, New features for duplicate bug detection, in: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, Association for Computing Machinery, New York, NY, USA, 2014, p. 324–327. doi:10.1145/2597073.2597090.
URL https://doi.org/10.1145/2597073.2597090

[28] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, Information Processing and Management 24 (5) (1988) 513–523. doi:https://doi.org/10.1016/0306-4573(88)90021-0.
URL https://www.sciencedirect.com/science/article/pii/0306457388900210

[29] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805.