Mining Software Profile across Multiple Repositories for Hierarchical Categorization

Tao Wang*, Huaimin Wang*, Gang Yin*, Charles X. Ling†, Xiang Li* and Peng Zou†

*National Laboratory for Parallel and Distributed Processing,
College of Computer, National University of Defense Technology, Changsha, 410073, China
taowang.2005@gmail.com, whm_w@163.com, {jack.nudt,shockleylee}@gmail.com
†Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7
cling@csd.uwo.ca
‡Academy of Equipment, Beijing, 100000, China
zpeng@nudt.edu.cn

Abstract—The large amounts of software repositories over the Internet are fundamentally changing the traditional paradigms of software maintenance. Efficient categorization of the massive projects for retrieving the relevant software in these repositories is of vital importance for Internet-based maintenance tasks such as solution searching, best practices learning and so on. Many previous works have been conducted on software categorization by mining source code or byte code, which are only verified on relatively small collections of projects with coarse-grained categories or clusters. However, Internet-based software maintenance requires finer-grained, more scalable and language-independent categorization approaches.

In this paper, we propose a novel approach to hierarchically categorize software projects based on their online profiles across multiple repositories. We design a SVM-based categorization framework to classify the massive number of software hierarchically. To improve the categorization performance, we aggregate different types of profile attributes from multiple repositories and design a weighted combination strategy which assigns greater weights to more important attributes. Extensive experiments are carried out on more than 18,000 projects across three repositories. The results show that our approach achieves significant improvements by using weighted combination, and the overall precision, recall and F-Measure can reach 71.41%, 65.60% and 68.38% in appropriate settings. Compared to the previous work, our approach presents competitive results with 123 finer-grained and multi-layered categories. In contrast to those using source code or byte code, our approach is more effective for large-scale and language-independent software categorization.

Keywords—Software Repository; Software Profile; Hierarchical Categorization

I. INTRODUCTION

The Internet-based software online repositories such as SourceForge, Ohloh and RubyForge1 hold large amounts of software projects, which create considerable opportunities for software engineering and are fundamentally changing the traditional paradigms of software maintenance. Traditional software maintenance tasks such as correcting discovered faults, adapting the system to changes in the environment, and improving the systems reliability and performance [1] are often confined to closed development teams or organizations and carried out in-house by using their internal repositories. With

their works solve the problem of categorizing software whose source code are not available.

In these works, most of them only experiment on relatively small collections of projects with flat and coarse-grained categories like “Internet” and “Games/Entertainment”. Such coarse-grained categories are not sufficient enough for retrieving the most related ones among the huge amounts of projects in the repositories. For example, in SourceForge there are more than 35,350 projects under the category “Internet”, which are too many for user to give a quick and feasible choice. Thus, finer-grained categorization is urgently needed. However, considering the large amounts and the complexity of the projects in these repositories (e.g. there are more than 400,000 projects in Ohloh which are programmed in more than 100 different languages), efficient and automatic categorization of these large-scale repositories is quite a challenging problem.

In this paper, we propose a hierarchical categorization approach which leverages the software online profiles as source information to do categorization. Online software repositories often curate a profile for each project, which summarizes the resource. A profile mainly consists of several types of attributes, i.e., a piece of description, a set of tags, and so on. The profiles cover the functional or technical aspects of the software, which make them effective and alternative source information for categorization. Taking the popular database management system MySQL as an example, it is described and categorized/tagged as shown in Table I and Table II in different repositories.

### Table I. Descriptions of MySQL in different repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Software Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sourceforge</td>
<td>MySQL is a well-known relational database manager used in a wide variety of systems, including... MySQL is a good choice for any situation requiring a database.</td>
</tr>
<tr>
<td>Ohloh</td>
<td>MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.</td>
</tr>
<tr>
<td>Freecode</td>
<td>MySQL is a widely used and fast SQL database server. It is a client/server implementation that consists of a server daemon (mysqld) and many different client programs/libraries.</td>
</tr>
</tbody>
</table>

### Table II. Categories/Tags of MySQL in different repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Software Categories/Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sourceforge</td>
<td>Database, Engines/Servers</td>
</tr>
<tr>
<td>Ohloh</td>
<td>db, acid, database_server, ndbms, database, db, jdb, actions, sql, software_development, mysql, dbms, ...</td>
</tr>
<tr>
<td>Freecode</td>
<td>Database, Database Engines/Servers</td>
</tr>
</tbody>
</table>

The two types of profile attributes cover the important features of MySQL as shown in the Table I and II. Different from the identifiers or APIs in source code which usually reflect the detailed and implementation-level features of classes or packages, software profiles emphasize on the high-level functional or technical features of the whole software. These online profiles are given in natural language and widely used as normal software documents in repositories, they are more attainable and can be used for categorization regardless of the software programming languages.

Our approach first constructs a category hierarchy which contains more than 120 categories organized in four levels based on the predefined categories in SourceForge. Then we design an SVM-based (Support Vector Machine) categorization approach, which classifies software hierarchically based on the software online profiles. We present the preliminary results and further analysis on how to improve the categorization performance by profile aggregation. Specifically, we design a weighted combination strategy to assign greater weights to important profile attributes and significantly improve the categorization performance. The main contributions of this paper include:

- We propose a hierarchical software categorization framework. It classifies software into multi-grained categories hierarchically. To the best of our knowledge, this is the first work on automatic software categorization hierarchically and the number of categories is significantly enlarged compared to previous works.
- We explore the multiple types of attributes in software online profiles for categorization and design an efficient combination strategy to aggregate them from multiple repositories. Such web-based software data is less studied for categorization in the previous works.
- We conduct extensive experiments on more than 18,000 software. The experiments show promising results for hierarchical categorization and prove the effectiveness and scalability of our method.

The reminder of this paper is organized as follows. Section II discusses the related works on software categorization and profile mining. Section III describes the hierarchical categorization approach in detail. Section IV presents the experiment questions and settings and Section V evaluates our approach. We discuss the validity of our work in Section VI, summarize the paper and discuss the future work in Section VII.

## II. Reviews of Precious Works

In this section, we review the previous works on software categorizations and software profiles mining.

### A. Software Categorization

Many works have been conducted on software categorization. According to the source information used for analysis, these works can be mainly classified into two groups.

The first is about categorization based on source code identifiers and comments. [13], [11], [12], [17] are the typical works that leverage source code information to do categorization. In most of these works, software projects are viewed as documents consisting of source code identifiers and comments. Textual classification approaches are employed to categorize. The main differences among these works are the document representation methods and the machine learning techniques used for categorization. [13] extracts source code identifiers and comments as documents, then applies an EEL[18] approach to do feature selection, and makes use of SVM approach to categorize the software into predefined topic and language categories. MUDABlue [11] and LACT [12] first generate categories from source code and then use LSA and LDA approaches separately to classify software. [19] enhances LACT by integrating domain knowledge for Android applications into the original approach and improves the performance.
The second group is software categorization based on other information like API calls [20], [15]. As the source code of many commercial software are not available, McMillan et al. [20] proposed a new categorization approach based on software API calls. The basic idea is that: external APIs and methods in software are grouped by their functions, thus they can be indicators of categories for the software that use these APIs.

Most of these works categorize software based on the source code or byte code information. Considering the scale of the repositories and the complexity of software source code or byte code, such approaches are not scalable for repository-scale categorization. In this work we explore the capability of software online profile in multiple repositories for scalable and efficient categorization.

B. Software Online Profile Mining

As more and more software projects are published with profiles, many researchers pay attention to such online data for different aims and get valuable results.

Dumitru et. al [4] study the software online feature descriptions in Softpedia to assist domain analysis. They propose an incremental diffusive clustering algorithm to discover domain-specific features in the massive amounts of software descriptions, and then recommend features for domain analysts after an initial input is provided. McMillan et. al [21] go a step further. By locating chains of function invocations in the source code, they correlate the mined features with corresponding implementation modules.

Recently, tags are widely used to describe software features at repositories. David Lo et. al study the value of software collaborative tags for different aims. In [22] they make use of co-occurrence of tags in Freecode to measure the similarities between pairs of tags and propose a $k$-medoids clustering algorithm to construct taxonomy of tags. In addition, they explore the software categories, license, programming languages and other related tags in SourceForge to find similar applications [23].

These works study the new types of software information, but they analyze these information independently and focus on one single repository. Differently, we aggregate the software descriptions and collaborative tags across multiple repositories for hierarchical categorization.

III. OUR APPROACH

In this section, we first design a hierarchical categorization framework and then discuss the software online profiles for the categorization.

A. Hierarchical Categorization Framework

1) Category Hierarchy Definition: As there are more than one million projects over the Internet, coarse-grained categorization is not efficient. We propose a hierarchical categorization system which includes multi-grained categories that are organized into a hierarchical structure. The category hierarchy is built according to the topics the category covers and the relations among them. For example, in SourceForge the coarse-grained category “Multimedia” is divided into several more specific subcategories like “Video”, “MP3” and so on.

In practice, a software category may belong to two or more parent categories. To simplify the problem, in this paper we model the software category hierarchy as a tree. In this tree, the relation between a pair of child and parent means “IS-A”. It has the following constrains of Asymmetric, Anti-reflexive and Transitive. Figure 1 presents a simple demonstration of the category hierarchy. The “Multimedia” is the root of this category tree, which covers all the other more specific categories like “Video”, “Sound/Audio” and so on. The categories “Video” and alike will form corresponding sub-trees. In this tree, each subcategory can have zero, one or several subcategories, but can only have one direct parent as constrained above. In this paper, we design a virtual category “Root” which is directly connected with all the first-level categories like “Multimedia” here to form a single tree.

Because it is difficult to define a complete category hierarchy to cover all possible fine-grained topics, some software may belong to none of the given leaf categories but an internal one. It means that the most specific categories for software can be internal or leaf categories in the hierarchy. In this hierarchy, a category is allowed to have one single subcategory, just like the node “Players” in Figure 1. Because it is not mandatory for projects to be assigned with leaf categories, this setting makes sense. This is motivated by the requirement to category software as specific as possible. For projects categorized under “Players”, they will be further tested to see if can be assigned with the more specific “MP3”.

2) Hierarchy Categorization Learning: There are many ways to build the hierarchical categorization model, which can be mainly classified as big-bang approach and top-down approach [24]. Big-bang approach treats all the categories at once and learns a single classifier for the entire hierarchy. Such approach will reduce the total size of classification model considerably. However, as the total number of categories is often large, it is difficult to build a single accurate classifier. Top-down approach adopts a different strategy. It builds multiple local classifiers and predicts each subcategory in the hierarchy separately. In [25] it has proved that the top-down approach is superior to the big-bang approach. Thus, in this paper we adopt a top-down approach for training the model. We build local binary classifier per node in the tree except the “Root”, which is the mostly used approach in literature to construct the hierarchical categorization model [24].

To build the local classifier for each node in category hierarchy, there are two key issues to consider. The first is what classification approach to use and the second is how to
select the positive and negative examples for training. For the first issue, there are quite many classification approaches to build the classifier like SVM, kNN, Naïve Bayes and so on. In this paper we choose SVM as our basis classifier because it has been proved superior to others for software categorization repeatedly in previous works [15], [26].

For the second issue, we adopt a “sibling” strategy in our framework. For a given category node in the hierarchy, the examples of both the node and its descendants are viewed as positive examples, and those categorized with its siblings as well as the descendants of the siblings are viewed as negative ones. For those categories that have no siblings (this is possible as defined in III-A1), the negative examples consists of those that categorized with the siblings of its nearest ancestor. For example, in Figure 1 the category “MP3” has no sibling, so its negative examples are those labelled with “CD Audio” which is the sibling category of its parent. Such strategy is adopted to avoid the serious imbalance problem [25].

3) Hierarchical Categorization Prediction: The hierarchical software categorization problem is a non-mandatory leaf node prediction problem [27]. It means that the most specific predicted category for a testing software can be any node (i.e. internal or leaf node) in the category hierarchy except the “Root”. For example, because categories “Video Capture” and “Display” together may not cover the whole topics of “Video”, a project may belong to none of categories “Video Capture” and “Display” but their parent category “Vedio”.

For a given software, we still take a top-down approach to predict its categories. We first test the given software over all the first-level categories with corresponding local classifiers. Then we only go down to test those subcategories whose parents have been predicted positive. One thing we need to note is that one software may belong to two or more categories of the same level in practice, which is a multi-label classification problem [28]. As we adopt binary SVM as our basic local classifier which predicts the categories of the same level separately, the multi-label problem can be solved naturally.

B. Software Profiles for Hierarchical Categorization

Different from the previous works which make use of source code or byte-code information, we leverage the software online profiles for categorization in this paper. As an alternative source information, they are less studied before.

1) Software Online Profiles in Repositories: In this paper, we mainly focus on two types of software profile attributes: software descriptions and collaborative tags. The software resources online are often published with brief descriptions which give high-level summarizations. For example, MySQL is hosted in several large software repositories and Table I in Section I lists its descriptions in SourceForge, Ohloh, Freecode. From this table we can observe that the descriptions of MySQL in different communities are different, each highlights some features of the software. The combination of these descriptions will give a more comprehensive summarization of it. In the descriptions, the terms like “SQL”, “database management system”, “Oracle”, which often appear more frequently in database related software, suggest the category of this software.

In addition to the high-level descriptions, collaborative tagging is widely used in software repositories to annotate resources. For example, in Ohloh and Freecode, all registered users are allowed to label the resources in the repository based on their experiments and knowledge. Table II shows the tags of MySQL in Ohloh and Freecode. These annotations aggregate the crowds understanding and reflect the features of the resources, which provide useful information for software categorization [23]. In specific, these tags mainly consists of two types. The first type is about functional topics of the system, such as “database, software development” for MySQL. The second type is about the techniques features of the project like “odbc, sql, jdbc”. For the functional topics or tags, they often explicitly reflect the category of the software. While for technique words or tags, as some techniques tend to be used more often in some specific categories of software, such tags are effective indicators of the system’s category.

The software profiles present high-level and important features of the resource, which is complementary to the detailed source code or API calls. Source code identifiers or API names often reflect detailed features of the software at a granularity of method, class or package. While the software descriptions and tags provide high-level summarizations of functional or technical features over the whole software. This is the key motivation for us to explore the capability of these two types of software profile attributes for categorization.

2) Combination of Software Profiles for Categorization: Although software descriptions and tags are both high-level summaries of the resources, there are some differences between them. Software descriptions are often given by software managers to exhibit and popularize the resource. There are many words that are not related to software functions or techniques, which are often noise for categorization. Differently, collaborative tags are often labelled by software users, maintainers or other software stockholders to annotate the key features of the resource. These tags covers the functional or technical aspects of the resource. Thus, they are often of better quality except some idiosyncratic or misspelling ones. Such differences are obvious as shown in Table I and II. These two types of profile attributes are complementary to each other and the combination of them will give more comprehensive summarization. However, because most of the projects are annotated with much less tags compared to the number of words in the descriptions, direct combination of them will dilute the weights of tags.

To balance the impacts of tags and descriptions on categorization, we distinguish the tags from the common words in descriptions and design a weighted combination strategy to strengthen the weights of tags. We duplicate the tags several times before combining them with software description. The duplicate time is decided according to the ratio between the length of software descriptions and its tags. The intuition behind this is as follow. As annotated by crowds, collaborative tags present the key features of the software, and they should be equivalent to software description at summarizing the software. Thus, the overall normalized term frequency of all the tags for a software should be proximity to that of the description. So we repeat the tags many times to make the total length of them be proximate to that of software description. For tags from different repositories, as we will take steps to only retain the commonly used ones as discussed in the experiments, currently the sources of these tags are not taken
into consideration for deciding their weights. The duplication times $\delta$ is decided according to Eq. 1.

$$\delta = \alpha \times \sqrt{\frac{\sum_k t_{kj}}{\sum_m t_{mj}}}$$ (1)

In Eq. 1 $t_{kj}$ represents the appearance number of term $k$ in project $j$ and $t_{mj}$ is that of tag appears in project $j$. In practice, some software are only annotated with very few tags which fail to cover all aspects of what description presented. To reduce the influence of these tags in such software, we take the square root over the ratio. In addition, we multiply it by another parameter $\alpha$ to control the overall duplication times. When $\alpha$ is set to 0, it becomes the simple combination without duplication which views tags as common words in descriptions.

In this paper, we use the traditional TF-IDF to represent the importance of a term for distinguishing a project at categorization. The traditional TF-IDF is computed for each word/tag according to Eq. 2. In this Equation, $t_{ij}$ represents the count of appearance for term $i$ which can be a common word in description or a tag for software $j$. $n_i$ stands for the number of software in which term $i$ appears and $N$ is the total number of software in the corpus. The first part of the equation is the normalized term frequency of word $i$ in project $j$, and the second part is the inverse document frequency of word $i$.

$$w_{ij} = \frac{t_{ij}}{\sum_k t_{kj}} \times \log \frac{N}{n_i}$$ (2)

According to Eq. 2, the duplication of tags will increase of normalized term frequency of the tags and decrease the term frequency of the words in description. Because it will not change the the inverse document frequency, the duplication of tags will increase the TF-IDF weights of tags and decrease that of description words.

IV. EXPERIMENTS DESIGN

In this section, we describe the experiment questions, experiment dataset and settings as well as the corresponding evaluation metrics.

A. Experiment Questions

To explore the effectiveness of different software profile attributes for categorization and comprehensively evaluate the proposed hierarchical categorization framework, we focus on the following four experiment questions.

- **Q1**: Which type of profile attributes is more effective for categorization, software descriptions or collaborative tags?
- **Q2**: What is the detailed performance of our approach for different level of categories?
- **Q3**: Are software online profiles as effective as API calls extracted from source code and byte code for categorization?
- **Q4**: Is profile-based categorization scalable for categorizing Internet-scale software repositories?

For experiment question Q1, we explore the effectiveness of the two types of profile attributes for categorization: software descriptions and collaborative tags. For Q2 we have a insight look at the the performance of our approach at different category levels. For Q3 we aim to compare the capability of software profiles and API calls for categorization. For Q4, we analyse the capability of our approach for scalable categorization.

B. Dataset and Experimental Settings

To address the above research questions and validate our approach, we focus on three large and popular open source repositories: SourceForge, Ohloh and Freecode. In this subsection we present the experiment dataset and the constructed category hierarchy.

1) Experiment Dataset: SourceForge is one of the largest and most popular open source community. It has predefined a hierarchical category system with 363 categories which is the basis for constructing our category hierarchy, and these projects that are categorized can be our training samples. Ohloh and Freecode are both large open source repositories which collect more than 400,000 and 45,000 software project respectively. In Ohloh and Freecode, they adopt a collaborative tagging mechanism to annotate and organise these collected resources. We crawl project profiles from Ohloh and Freecode to enrich the profile of software in SourceForge.

We construct the experiment dataset through the following steps: (1) **Software homepage crawling**. We crawl the homepage of the software in the three open source repositories and parse them to extract the profile attributes including descriptions, categories and tags. (2) **Software profile aggregation**. Among these crawled software projects, a proportion of them exist in multiple repositories, like MySQL listed in Table I. Such software not only have categories in SourceForge but also have tags in Ohloh or Freecode. We retain these projects and combine their profiles to train hierarchical classifiers. (3) **Profile preprocessing**. We do stop words removing and stemming over the descriptions and tags. After this, we only retain those projects which have combined descriptions of more than 10 words. In addition, to get rid of those idiosyncratic or misspelling tags, we set a threshold of 50 to filter such tags, i.e. only those tags that have been used more than 50 times in the corresponding repositories will be retained. After the three steps, we get a number of 18,032 unique software and a total of 5,429 unique tags. The detailed information of the dataset is shown in Table III.

The first part is the projects from SourceForge, each of which has at least one category. These software projects have descriptions of about 20 words and 3 categories in average; The second and third part are the projects from Ohloh and Freecode. There are 9,813 and 10,357 projects in Ohloh and Freecode which also exist in the 18,032 SourceForge projects. The total projects in Ohloh and Freecode are 2,138 more than that of SourceForge which implies that there are 2,138 projects among the dataset exist in all the three repositories. These software in Ohloh and Freecode have similar description length and average tags. The retained projects only account for a small proportion of the total software in these repositories. There are several reasons. Firstly, the software projects that
are categorized in SourceForge and also tagged in Ohloh or Freecode take only a small proportion [29]. Secondly, we take a strict matching strategy on full-name to search projects in multiple repositories for profile aggregation. Such strategy is adopt to avoid those projects which have similar name but actually not the same.

2) Category Hierarchy Construction: We build the category hierarchy based on the predefined one in SourceForge. The categories in SourceForge include coarse-grained ones like “Multimedia” and “Games” as well as specific ones like “MP3” and “First Person Shooter”, which are organized in four levels. Similar categorization systems are also adopted in other communities like Rubyforge. Here we constructed our category hierarchy based on the SourceForge categorization system as follows.

1) Transforming the DAG structure to tree structure. In the original category hierarchy, there can be more than one path between two nodes. To simplify the classification process, we only retain the mostly used path in the dataset.

2) Pruning the category hierarchy. Firstly we combine those categories which are quite similar. Secondly we delete those categories which have less examples than a predefined threshold and then we lift the examples under them to the corresponding parent categories.

3) Constructing the uniform category hierarchy. After DAG transformation and hierarchy pruning, we create a virtual root to cover all the original root categories. This results in a single category hierarchy tree.

For category pruning, the thresholds for deletion are set to 500, 100, 50, 50 for the four levels categories. After the preprocessing, we constructed a uniform hierarchy which consists of 123 categories with four levels. Figure 1 presents a part of the constructed category hierarchy under “Multimedia”.

The summarization of the constructed hierarchy is presented in the Table. IV. There are 12 top categories at level one including “Multimedia, Games/Entertainment, Office/Business” and so on. Most of the categories are cramped in the second and third level, which have 61 and 41 of them respectively. The fourth level have only 9 categories. Among the 18,032 projects in the dataset, the average number of positive samples for the first level categories is 2215.33. As the category level increases, the average number of positive samples decreases dramatically to only a few hundreds, and this will affect the performance of our approach. More samples will be included in the future.

C. Evaluation Metrics

In previous works like [15], [20], the precision, recall and F-Measure are widely used to evaluate the performance of flat categorization system. In this paper we make a slight change on these metrics and adopt hierarchical metrics including hierarchical precision(hP), hierarchical recall(hR) and hierarchical f-measure (hF) over each category. The definition of these metrics are as follows.

\[
hP = \frac{\hat{P}_i \cap \hat{T}_i}{\hat{P}_i}, \quad hR = \frac{\hat{P}_i \cap \hat{T}_i}{T_i}, \quad hF = \frac{2 \cdot hP \cdot hR}{hP + hR}
\] (3)

In Eq. 3, for each category \(i\), \(\hat{P}_i\) is the predicted sample set for category \(i\). It consists of software whose categories are predicted as category \(i\) or the descendants of \(i\), \(\hat{T}_i\) is the true sample set that consists of all projects labelled with category \(i\) and the descendants of \(i\). The three metrics \(hP, hR\) and \(hF\) represent the average precision, recall and F-Measure for each category over all the testing examples where the hierarchical structure is concerned.

To measure the average performance over all the categories, we make use of Micro Average on hierarchical Precision(Micro-hP), Recall(Micro-hR) and F-Measure(Micro-hF) as Eq. 4 which are similar to these used in [24], [30].

\[
\begin{align*}
\text{Micro-hP} &= \frac{\sum\limits_i (\hat{P}_i \cap \hat{T}_i)}{\sum\limits_i \hat{P}_i}, \\
\text{Micro-hR} &= \frac{\sum\limits_i (\hat{P}_i \cap \hat{T}_i)}{\sum\limits_i T_i}, \\
\text{Micro-hF} &= \frac{2 \cdot \text{Micro-hP} \cdot \text{Micro-hR}}{\text{Micro-hP} + \text{Micro-hR}}
\end{align*}
\] (4)

V. EXPERIMENT EVALUATIONS

In the experiments, we first build the hierarchical categorization model and then test on large amounts of projects. The SVM classification algorithm is used as the basic classifier and the parameters are set as default except the parameter \(c\). In the experiments we set \(c\) to 0.5 for which achieves the best performance after a simple test. The detailed information of the datasets used for experiments and the hierarchical categorization framework are discussed in Section IV. We first use 5-fold cross validation and hierarchical metrics to measure our approach and compare the different types of profile attributes for categorization. Then we compare our approach with a previous work which categories software based on API calls. For 5-fold cross validation, the whole dataset is broken into 5 folds randomly, and each of the folds will be tested once by using the other 4 folds to train the model.
A. Q1: Software Descriptions and Collaborative Tags for Categorization

We explore two different types of profile attributes in this work: software descriptions and collaborative tags. These attributes have never been extensively studied for software categorization before. In this section we conduct four sets of experiments to compare the effectiveness of the different types of profiles for categorization. Firstly we only make use of SourceForge software descriptions to train and test. Then we aggregate the software descriptions from the three repositories to test. Thirdly, we do experiment on combination of software tags from Ohloh and Freecode. Finally, we simply combine software descriptions with tags from all these three repositories to do categorization.

Table V shows the experiment results. SF descriptions, Combination of descriptions, Combination of tags and Combination of descriptions and tags in the table stand for the four sets of corresponding experiments mentioned above where SF descriptions stands for the experiment based on SourceForge software descriptions. In this experiment, we get an overall micro precision of about 58.68% and recall of 48.37%. By aggregating the descriptions from other repositories, the categorization precision is only improved slightly while the recall is almost the same. This is because that for a large proportion of software, the descriptions of the same software in different repositories overlap each other a lot, which fails to provide much additional information. For collaborative tags, though each software project has much less tags than description words, the experiment based on combination of tags still achieves better results. The micro precision is similar to that of Combination of description, but the micro recall shows a significant improvement of about 13.68% and F-Measure gets an improvement of 8.50%. This is due to the better quality of collaborative tags. Different from software descriptions which often contains many words like “wide range”, “performance” for MySQL that reflect no technical or functional features, most of the tags have specific meaning and reflect some aspects of features for the resource. Thus, categorization based on collaborative tags achieves better results.

As shown in the fourth row, we simply combine the software descriptions and tags. In this experiment we get better precision but worse recall than that of tags. Compared to collaborative tags, such combination treats each tag as common word in description. On the one hand, such combination aggregates more information of the software; On the other hand, it introduces more noisy words and dilutes the weights of tags. Thus, it fails to improve the categorization performance over that of Combination of tags.

<table>
<thead>
<tr>
<th>Software profile</th>
<th>Micro-hP</th>
<th>Micro-hr</th>
<th>Micro-hF</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF descriptions</td>
<td>0.5368</td>
<td>0.4837</td>
<td>0.5302</td>
</tr>
<tr>
<td>Combination of descriptions</td>
<td>0.6213</td>
<td>0.4839</td>
<td>0.5440</td>
</tr>
<tr>
<td>Combination of tags</td>
<td>0.6375</td>
<td>0.6207</td>
<td>0.6290</td>
</tr>
<tr>
<td>Combination of descriptions and tags</td>
<td>0.6831</td>
<td>0.5746</td>
<td>0.6241</td>
</tr>
</tbody>
</table>

To make full use of the descriptions and tags for categorization, in Section III-B2 we propose a weighted combination strategy which assigns more weights to more important attributes by duplication. In Eq. 1 we design a parameter $\alpha$ which controls the overall duplication time for tags. Figure 2 presents the performances for different values of $\alpha$.

![Micro-Hierarchical Precision with different $\alpha$](image1)

![Micro-Hierarchical Recall with different $\alpha$](image2)

![Micro-Hierarchical F-Measure with different $\alpha$](image3)

Fig. 2. Categorization performance with different values of $\alpha$ based on weighted combination of profiles

It shows that as the $\alpha$ increases from 0 to 1, both precision and recall have a great improvement, and the F-Measure improves from about 62.41% to about 68.38% . This verifies the effectiveness of our weighted combination strategy. While the parameter $\alpha$ keeps on increasing from 1 to 3 and larger, the precision decreases slowly, and the recall and F-Measure keep almost the same. This is reasonable. As $\alpha$ increases, the weights of tags will become much larger than that of words in descriptions. This leads to the result that the descriptions have only slight effect on distinguishing software categories.

Overall, from the above results we can find that collaborative tags are more effective attributes than software descriptions for categorization. Nevertheless, those two types of attributes are mutually complementary, and the weighted combination of them which assigns greater weights to tags properly will improve the overall performance. In addition, from Table V we can see that, even without the additional tags and descriptions, our hierarchical categorization framework is still applicable by only using the descriptions from SourceForge and achieves an overall F-measure of about 53.02%.
B. Q2: The categorization performance for different category levels

In Q1 we present the average results over the whole four-level category hierarchy. As different level of the categories often have different number of positive examples, it will affect the performances of categorization. To get a comprehensive understanding of such influence, we analyze the performance for the top-k level categories where k ranges from 1 to 4. Table VI presents the detailed results based on weighted combination of profiles with \( \alpha = 1 \) as discussed in Section V-A.

<table>
<thead>
<tr>
<th>top-k</th>
<th>Num. Categories</th>
<th>Micro-hF</th>
<th>Micro-hR</th>
<th>Micro-hP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>0.8534</td>
<td>0.7908</td>
<td>0.7695</td>
</tr>
<tr>
<td>2</td>
<td>73</td>
<td>0.7767</td>
<td>0.6647</td>
<td>0.7163</td>
</tr>
<tr>
<td>3</td>
<td>114</td>
<td>0.7365</td>
<td>0.6567</td>
<td>0.6942</td>
</tr>
<tr>
<td>4</td>
<td>123</td>
<td>0.7141</td>
<td>0.6560</td>
<td>0.6838</td>
</tr>
</tbody>
</table>

As shown in this table, the overall performance of the first level of 12 categories achieves 85.34%, 70.08% and 76.95% for precision, recall and F-measure respectively, which is relatively high. As the category level rises, the overall performance decreases accordingly. This is because the deeper categories often have fewer examples which affect the categorization accuracy. As listed in Table IV in Section IV-B2, the average number of positive examples decreases from 2,215 at first level to about 120 at fourth level. It will be our future work to introduce more samples for training more accurate categorization model.

C. Q3: Weighted Combination of Software Profiles and API calls for Categorization

In this section we carry out an experiment to compare the categorization performance by using software profiles and that of software API calls. APIs are grouped into packages and libraries according to their functions. Thus, the APIs invoked in a software would be good indicators of the software category. Based on this intuition, McMillan et. al [20] leverage API calls for categorization. Their case studies suggest that API calls are as effective as source code identifiers and comment terms for categorization. In their works the effectiveness of using API packages and API classes are verified respectively. We only compares their best setting (using API packages) with our approach by using profile combination with \( \alpha = 1 \).

In the experiment, we test our approach on the dataset used in their experiments. Among their SourceForge dataset, we search for the projects which are tagged in Ohloh or Freecode and get a total of 849 ones which are used as the testing set. Then we train the categorization model based on the our dataset in which the testing set are rejected. The final results for the 22 categories are shown in Table VII.

Overall, the two approaches achieve similar F-Measure over all the 22 categories as shown in the last row of the table. In specific, for some categories like “Chat”, “Communications” and “Emails”, our approach get better results. While for some others such as “Visualization” and “Front-Ends”, API-based approach achieves higher F-Measure. Such difference maybe result from the different characteristics of attributes used for categorization. The name of API packages and classes are implementation-related attributes which mainly reflect the functionality of the package or the class, which is fine-grained features. While the software profiles emphasis on high-level summaries of the resource. These two types of attributes reflect the features of the software from different perspectives with different granularity. The results suggest that the software profiles are effective alternative to software APIs for categorization. Intuitively, they are complementary to each other and a proper integration of them should be more effective. It will be our future work to have a comprehensive analysis on this.

D. Q4: Scalability Analysis

It should be noted that we do categorization hierarchically with more finer-grained categories. In addition to the 22 categories, some of the 849 testing software in Sourceforge are originally classified with many finer-grained categories. Our approach organizes these categories hierarchically and achieves high accuracy at these specific categories as well. For example, in our constructed category hierarchy, the category “Compilers, Testing” in the 22 categories are organized under the category “Software Development”. The “Software Development” category are divided into more specific ones like “Build Tools, Object Oriented, Algorithms, Quality Assurance” and so on. Among these testing software, many of them are originally grouped under these specific categories, and our approach classes these with high accuracy. Table VIII presents the results for the categories under “Software Development”.

<table>
<thead>
<tr>
<th>Category</th>
<th>API calls</th>
<th>Software profiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bio-Informatics</td>
<td>0.743</td>
<td>0.583</td>
</tr>
<tr>
<td>Chat</td>
<td>0.742</td>
<td>0.815</td>
</tr>
<tr>
<td>Communications</td>
<td>0.601</td>
<td>0.723</td>
</tr>
<tr>
<td>Compilers</td>
<td>0.712</td>
<td>0.616</td>
</tr>
<tr>
<td>Database</td>
<td>0.689</td>
<td>0.664</td>
</tr>
<tr>
<td>Education</td>
<td>0.633</td>
<td>0.583</td>
</tr>
<tr>
<td>Email</td>
<td>0.746</td>
<td>0.792</td>
</tr>
<tr>
<td>Frameworks</td>
<td>0.727</td>
<td>0.806</td>
</tr>
<tr>
<td>Front-Ends</td>
<td>0.702</td>
<td>0.457</td>
</tr>
<tr>
<td>Games/Entertainments</td>
<td>0.728</td>
<td>0.703</td>
</tr>
<tr>
<td>Graphics</td>
<td>0.607</td>
<td>0.580</td>
</tr>
<tr>
<td>Indexing/Searching</td>
<td>0.732</td>
<td>0.667</td>
</tr>
<tr>
<td>Internet</td>
<td>0.604</td>
<td>0.581</td>
</tr>
<tr>
<td>Interpreters</td>
<td>0.635</td>
<td>0.679</td>
</tr>
<tr>
<td>Mathematics</td>
<td>0.658</td>
<td>0.775</td>
</tr>
<tr>
<td>Networking</td>
<td>0.567</td>
<td>0.481</td>
</tr>
<tr>
<td>Office/Business</td>
<td>0.598</td>
<td>0.617</td>
</tr>
<tr>
<td>Scientific</td>
<td>0.654</td>
<td>0.708</td>
</tr>
<tr>
<td>Security</td>
<td>0.622</td>
<td>0.727</td>
</tr>
<tr>
<td>Testing</td>
<td>0.730</td>
<td>0.678</td>
</tr>
<tr>
<td>Visualization</td>
<td>0.605</td>
<td>0.400</td>
</tr>
<tr>
<td>WWW/HTTP</td>
<td>0.696</td>
<td>0.605</td>
</tr>
<tr>
<td>AVG.</td>
<td>0.6696</td>
<td>0.6464</td>
</tr>
</tbody>
</table>
In this section we analyze the scalability of our approach for categorizing the Internet-scale repositories.

The cost of software categorization mainly consists of two parts. The first is training data obtaining and preprocessing and the second is the model training and prediction. For the first part, to get the profiles of the 417,344 software in Ohloh, we design a processing chain including web crawler and profile extractor. The crawler is implemented with 50 threads to crawl the homepage of the software and the profile extractor parses the homepage to get corresponding profile information. The software profiles are given in natural language which is not specific to any programming language. Thus, the profile extraction and pre-processing is simple. We deploy the crawler and extractor in a server (8*2.13G CPUs, 16GB RAM and 2TB storage) which is connected to the Internet with network bandwidth of 100M. Totally, it takes less than 3 days to get all the software profiles in Ohloh with a total size of about 100 MB for the extracted profile attributes.

The model training and category prediction on the pre-processed dataset are quite efficient. The cost time for model training and testing mainly depends on the classification algorithm used. We do our experiments using SVM algorithm on a computer of Intel(R) Core(TM) i5-3320M CPU @ 2.6M with 4GB RAM. The total time for five-cross validation over 18,032 projects is about 620 seconds, in which the training time is about 540 seconds and testing time is about 80 seconds. Based on this we can estimate that the time for categorizing the total 417,344 software in Ohloh will take about 31 minutes once the categorization model is built.

In contrast, categorizing the repository software based on source code will be much more complex. Firstly, considering the huge amounts of projects in the repositories, to get the source code of all the software is time-consuming. Secondly, the analysis of the source code is quite complex. For example, in June 2011 MySQL has about 1,333,855 lines of code and 298,918 lines of comments\(^2\). To parse the source code of it for identifiers and comments is quite a time-consuming process, let alone the huge amounts of software which are implemented in various programming languages.

Based on the analysis we can conclude that our approach is capable of doing categorization for the large repositories with high efficiency.

VI. VALIDITY

There are some threats to validity which may affect the experiment results of our approach. One is that the software may be incorrectly categorized in SourceForge or incorrectly tagged in Ohloh and Freecode. It is very difficult to eliminate such kind of threat. In our paper, we minimize such threat by including as many samples as possible to reduce the ratio of such incorrectly categorized software.

Another threat is that the same name software we crawled from multiple repositories may not be the same software. Such mistakes will introduce incorrect profiles in the dataset. To minimise such threat, in this paper we adopt a strict full-name matching strategy to filter the false ones. In the future we will design a profile-comparison tool to compute the similarities of the software projects with same names, and thus the quality of training data will be further improved.

As discussed in Q1, the software tags have a great impact on the performance of our categorization approach. Thus, the tag shortage may affect the generality of our approach. However, for such software that have no tag, we can classify them based on their descriptions. As shown in Table V, the categorization precision based on descriptions achieves 58.68% as well. In addition, as social tagging is becoming widely accepted, more and more software will be annotated with tags, which can be re-categorized efficiently based on the new data.

VII. CONCLUSION AND FUTURE WORKS

Internet-scale software repositories requires more scalable and finer-grained categorization. In this paper we propose a hierarchical categorization approach by mining software online profiles across multiple repositories. We categorize software with more than 120 categories which are organized into a hierarchical structured. Such categorization greatly improve the efficiency for retrieving similar software. As online profiles is not specific to any programming languages or source code, and it is easy to obtain and analyse as well. Thus, it is scalable to do categorization over the huge amounts of software in repositories efficiently. The extensive experiments on more than 18,000 software show that our approach achieves promising performance over the whole category hierarchy. Compared to the approach that using APIs for flat categorization, our approach achieves competitive results, and furthermore, our approach is capable of predicting more specific categories hierarchically.

In addition, we have developed an open source software searching and ranking system named Influx\(^3\), which has crawled OSS data from the influential repositories (such as SourceForge, Ohloh, Freecode). Currently, Influx provides some interesting services by mining OSS data, such as cross-repositories profiles, synergy analysis and software ranking in some repositories (such as OW2). A demo of the hierarchical categorization system proposed in this paper has been integrated into Influx and can be visited now.

There are several possible ways to improve the performance for categorization. Firstly, more software can be crawled from websites over the Internet to enrich the training set. In this paper, although we crawled more than 18,000 software from three software repositories, the positive samples for some categories are only about 100, which affects the accuracy for classification. In the future, more samples can be retrieved from

\(^2\)https://www.ohloh.net/p/mysql

\(^3\)http://influx.trustie.net
various repositories to improve the performance. Secondly, in this paper we treat the tags from different repositories equally. Further analysis about the characteristics of these tags will be done to explore their potential for categorization. In addition, software attributes of API calls and software profiles reflect software features at different granularity, which are complementary to each other. We will study how to make full use of the strengths of these different attributes.

ACKNOWLEDGMENT

This research is supported by the National Science Foundation of China (Grant No.60903043) and the National High Technology Research and Development Program of China (Grant No. 2012AA011201). We would like to thank Xiao Li, Yue Yu and Li Fang for their collaboration. We would also thank the anonymous reviewers for providing us constructive comments and suggestions.

REFERENCES