2012 28th IEEE International Conference on Software Maintenance (ICSM)

Inferring Semantically Related Software Terms and Their Taxonomy By Leveraging Collaborative Tagging

Shaowei Wang, David Lo, and Lingxiao Jiang
School of Information Systems, Singapore Management University
{*shaoweiwang.2010,davidlo,lxjiang*}*@smu.edu.sg*

***Abstract*—Many software engineering tasks, such as feature location and duplicate bug report detection, leverages similarities among textual corpora. However, due to the different words used by developers to express the same concept, exact matching of words is insufficient. One document can contain a particular word while the other document may contain another word that is semantically related but is not the same. Such word differences may cause inaccuracies in subsequent software engineering tasks. Recently, tagging has impacted the software engineering community. Developers increasingly use tags to describe important features of a software product. Many project hosting sites allow users to tag various projects with their own words. It becomes increasingly important to understand and relate these tags.**

Based on the tags available from software project hosting websites, we propose a similarity metric to infer semantically related terms, each of which is a tag, and build a taxonomy that could further describe the relationships among these terms. We have built a sample taxonomy from tens of thousands of projects and their tags. Our user studies show that our proposed similarity metric for tags are indeed related to the semantic similarity of the terms, and the resultant semantic taxonomy among terms is reasonably good.

I. Introduction

Many software engineering tasks compare and contrast textual documents. For example, duplicate bug report detection leverages common terms appearing in two bug reports to decide if they are duplicates of each other [17]. Feature location finds pieces of code that implement a described functionality [15]. Since the same concept can often be expressed by various words that might either be synonyms or are semantically related, if only *common* words are searched for, certain semantically related documents described in different terms[1] might be missed. This problem has been termed as the *synonym problem* in the text retrieval community [4]. Thus, the accuracy of many software engineering tasks may also be improved if this synonym problem, instantiated in the software domain, is addressed .

For text retrieval, WordNet [1] has been widely used to detect synonyms [20]. However, software engineers often use jargons that either do not appear in WordNet or have different meanings from normal English. For example, words such as "bug" and "beetle" are semantically related in English, but are very different in software engineering context. Thus, there is a need to create a better WordNet—a domain specific one that addresses software engineering needs.

With the advent of Web 2.0, we have seen the proliferation of tagging on various items ranging from photos on Flickr, websites on Delicious, software systems on Freecode, etc. A tag is often used to indicate certain semantic aspect of related items, such as "Word Processor." We observe that such tags could be used to find semantically related software engineering terms and projects. For example, a project tagged with "document editor" may be semantically similar to a "word processor." With the increasing uses of tags, it can also be useful for software engineering purposes to establish a WordNet-like database for software terms.

In this work, we propose a new approach that detects similar software tags. We characterize a tag by the documents and document contents that it tags. We define a similarity metric between two tags by measuring the similarity among the documents. The similarity metric is in turn used together with a clustering algorithm in multiple iterations to build a hierarchy of tag clusters.

We have performed user studies to evaluate our approach. Given the similarity scores and the hierarchy of the tags, the users in general agree that the similarity metric correlates with the actual semantic relations among the tags and the tag hierarchy (i.e., taxonomy) is reasonable.

The contributions of this paper are as follows:

1) We propose a new problem of detecting semantically related software terms.
2) We propose a similarity metric among software tags by considering the common documents tagged by the tags.
3) We propose a solution based on the k-medoids clustering algorithm to create a taxonomy of tags.
4) Our user studies show that the similarity metric scores are closely related to the similarities in the semantic meanings of pairs of tags, and the taxonomy reasonably reflects the semantic relations among the tags.

The paper is organized as follows. Section II describes preliminary materials. Section III details our approach. Section IV presents empirical evaluation. Section V discusses related work. Section VI concludes with future work.

[1]We use "terms", "words", and "tags" interchangeably in this paper.

978-1-4673-2312-3/12/$31.00

II. Preliminaries

In this section, we describe preliminary information on software tagging, and the k-medoids clustering algorithm.

A. Tagging Software Engineering Data

Many project hosting sites, such as Freecode[2], allow developers to tag projects. On Freecode, information about more than one hundred thousands of applications is provided. Each application has the link for download, the description of the application, and tags indicating various features of the application. Sample project information from Freecode is shown in Figure 1. One can note that the Java Apple Computer Emulator is tagged with Major, Bug fixes, new features, LGPL, and computer emulator.

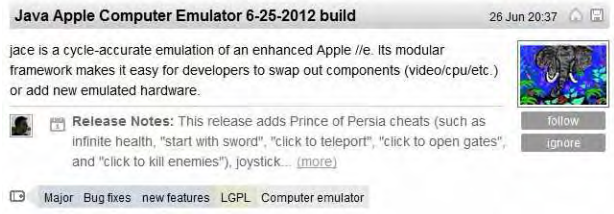

Figure 1. Project Information in Freecode

Users could create a Freecode account and provide information for an application. Freecode may be viewed as a Wiki-like platform for developers to share information about various applications. Such information provides a good knowledge base for us to infer semantically related software terms. In this study, we use the application descriptions and tags in Freecode for our purpose.

B. K-Medoids Clustering Algorithm

This algorithm splits a set of data points to a pre-set number (k) of groups (or clusters) so that the square error is minimized [10]. It is partitional, i.e., each data point is assigned to one and only one cluster. The algorithm requires a similarity metric between data points, and it performs many iterations as follows to decide the best way to split:

1) Randomly pick k points as cluster centers (medoids).
2) Assign each remaining data point (non-medoids) to the cluster whose medoid is closest. This would form a configuration (i.e., the initial k clusters).
3) Update the medoid for each cluster: choose the point in the cluster that has the minimum total distance to all other points as the new medoid.
4) Repeat steps 2-3 until no more change to the medoids.

III. Proposed Approach

Our approach mainly consists of two steps. First, we calculate the similarity between every pair of terms. To this end, we propose a similarity metric based on the documents that are tagged by the terms. Second, based on the similarity metric, we infer a taxonomy of the terms by repeatedly applying k-medoids clustering on the terms.

[2]http://freecode.com/

A. Calculation of Similarity Among Terms

Inspired by information retrieval techniques, we use the documents tagged by the terms to measure the similarity between terms (i.e., tags). In our setting, each document is a description of an application (cf. Figure 1). Two terms can be similar if they tag many *common documents*. We call this similarity the *document similarity* of two terms. In addition, we measure the similarity of two terms based on the textual contents of the documents tagged by them. We call this similarity the *textual similarity* of two terms.

Let $Doc(t)$ be the set of documents that are tagged with term t. We define the *document similarity* of two terms t_1 and t_2 as $dsim(t_1, t_2) = \frac{Doc(t_1) \bigcap Doc(t_2)}{Doc(t_1) \bigcup Doc(t_2)}$.

For *textual similarity*, we calculate it as the cosine similarity between the vectors representing the term frequencies & inverse document frequencies of the words appearing in the documents tagged by the terms as follows. First, for each term t, we apply standard text pre-processing techniques to $Doc(t)$ to remove common stop words, such as I, you, etc., and reduce words into their root forms by stemming (e.g., both reading and reads are reduced to read). The preprocessed set of documents is referred to as $Doc^P(t)$. Second, a vector, referred to as $V(t)$, is created: each element $V(t)[w]$ in $V(t)$ corresponds to one word w appearing in $Doc^P(t)$ together with its TF-IDF score (term frequency & inverse document frequency) [13]. Third, given two terms t_1 and t_2 and their word vectors $V(t_1)$ and $V(t_2)$, the textual similarity between t_1 and t_2 is calculated as the cosine similarity between $V(t_1)$ and $V(t_2)$:

$$tsim(t_1, t_2) = \frac{\sum_{w \in V(t_1) \cap V(t_2)} V(t_1)[w] \times V(t_2)[w]}{\sqrt{\sum_{w \in V(t_1)} V(t_1)[w]^2} \times \sqrt{\sum_{w \in V(t_2)} V(t_2)[w]^2}}$$

Finally, we combine document and textual similarities to derive the similarity score of terms t_1 and t_2. It is a weighted sum of both similarities, where we use 0.5 for w_1 and w_2:

$$sim(t_1, t_2) = w_1 \times dsim(t_1, t_2) + w_2 \times tsim(t_1, t_2) \quad (1)$$

B. Taxonomy Inference

We infer the taxonomy of software terms by performing repeated k-medoids clustering. At each application of k-medoids, we divide the set of terms into smaller groups. Each of these groups can then be divided into even smaller subgroups by applying k-medoids clustering again. K-medoids clustering requires a similarity metric between two data points (a data point is a term in our setting). This paper uses the similarity metric presented in Equation (1).

The pseudocode of the approach is given in Algorithm 1. The procedure `CreateTaxonomy` first initializes a dummy root node by setting its label as "root"; it then recursively applies k-medoids clustering at various levels in the taxonomy by calling `CreateLevels`. We stop dividing a cluster further if its size is less than $minSize$. In our evaluation, we use $minSize = 20$ and $k = 7$.

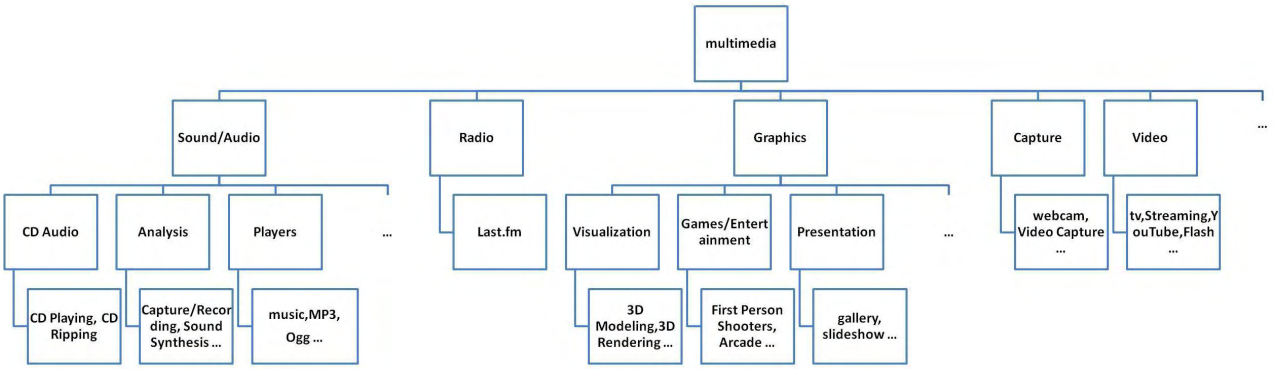

Figure 2. Partial resultant taxonomy

Algorithm 1 Construction of Term Taxonomy.

Procedure `CreateTaxonomy`
Input: $ipTerms$: Set of terms
k: # of sub-clusters under each node in the taxonomy
$minSize$: Minimum size of a cluster
Output: Taxonomy of terms in $ipTerms$
Initialize a dummy root node $root$ of the taxonomy
`CreateLevels`($ipTerms$, k, $minSize$, $root$)
Output the taxonomy rooted at $root$

Procedure `CreateLevels`
Input: $ipTerms$: Set of terms
k: # of sub-clusters under each node in the taxonomy
$minSize$: Minimum size of a cluster
p: Parent node in the taxonomy
Output: Children of p added to the taxonomy
Let $Clusters$ = Perform k-medoids on $ipTerms$
for all c in $Clusters$ **do**
  if $size(c) < minSize$ **then**
    Add c as a child of p in the taxonomy
  else
    Add the medoid m of c as a child of p
    Let $nTerms$ = All non-medoid terms in c
    CreateLevels($nTerms$, k, $minSize$, m)
  end if
end for

IV. Empirical Evaluation

In this section, we present our dataset, our research questions and answers, and describe some threats to validity.

A. Dataset

We collect 45,470 projects from Freecode with 7,163 tags, but remove tags used in less than 10 projects (they appear to be too specific) and projects with no tags. Thus, we get 690 tags and 40,744 projects.

B. Research Questions

We are interested in the following research questions:

- RQ1 How accurate is our similarity metric in measuring the semantic relations among terms?
- RQ2 How the resultant taxonomy looks like?
- RQ3 How accurate is the taxonomy? Is the structure of the taxonomy reasonable?

C. RQ1: Accuracy of the Similarity Metric

We perform a user study with six participants who all have background in computer science. We sort all pairs of terms based on their similarities calculated by the equations in Section III-A. The smaller the similarity is, the higher rank a pair gets. We pick 3 categories of pairs from the ranked list: 100 most similar terms (*top*), 100 most different ones (*bottom*), and 100 from the middle of the list (*middle*). Each participant is randomly assigned 50 pairs and asked to provide a 3-point Likert score to each pair to indicate whether they agree if the pair is similar: 3 means agree, 2 means neither agree nor disagree, and 1 means disagree.

Table I shows the statistics of Likert scores from the user study for each category. The mean Likert score for the top pairs is larger than that of the middle pairs, which is larger than that of the bottom pairs. Mann-Whitney U tests on the scores show that the differences are statistically significant, which means our similarity metric can indeed be an effective measurement of semantic relations among terms.

Table I
Likert scores for each category

Category	Mean	Deviation
Top	2.37	0.76
Middle	1.5	0.70
Bottom	1.03	0.17

D. RQ2: Resultant Taxonomy

Figure 2 shows a partial resultant taxonomy, which is a hierarchical tree. Each node except leaf nodes denotes a term (tag) which represents the general meaning of its children. Every leaf node is a cluster of terms. As Figure 2 shows, this partial hierarchy presents the tags under Multimedia, which can be categorized into Sound/Audio, Radio, Graphics, Capture, Video, etc. The resultant taxonomy structure reasonably represents the relationships among the terms.

E. RQ3: Accuracy of the Taxonomy

We also perform a user study with the same six participants. We randomly pick 100 paths from a non-leaf node in the taxonomy to a term in a leaf node. A sample path in Figure 2 is "Multimedia → Graphics → Games/Entertainment → First Person Shooters." Each participant is randomly assigned 20 paths and asked to provide a 3-point Likert score for each path to indicate whether they agree if the path is semantically reasonable: 3 means all terms in the path are

reasonable, 2 means some terms are reasonable, and 1 means all terms are unreasonable.

Figure 3 shows that 38% of the 100 paths are totally reasonable, 48% of them are partially reasonable, and only 15% of them are unreasonable. The mean Likert score is 2.23, which means that the taxonomy created by our approach is reasonably meaningful.

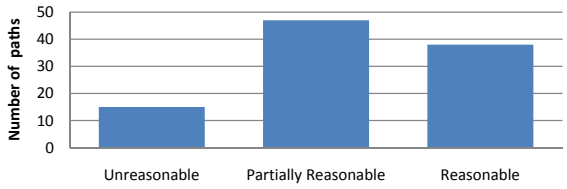

Figure 3. The histogram of Likert scores for the 100 paths.

F. Threats to Validity

There are threats due to our limited datasets, parameters used in our approach ($minSize$, k, etc.), randomness employed during user studies, and human bias in providing Likert scores. In the future, we plan to collect more tag data from various project hosting platforms, to perform sensitivity analysis of the parameters, and to involve more participants and more samples to reduce such threats.

V. Related Work

Study of Semantically Related Words: In software engineering communities, there are a few studies on semantically related words and their applications. Yang and Tan infer related words from software *code* (e.g., [22]). Biggers et al. consider a suite of similarity metrics based on *code* lexicons as well [5]. Kuhn uses word frequencies in *code* to automatically label software components [12]. Wang et al. infer paraphrases from duplicate bug reports [21]. Abebe et al. establish a catalog of "lexical bad smells" in code and use it to help concept localization [2], [3]. Haiduc and Marcus study how developers use domain terms in code (e.g., in identifiers or comments) [9]. Our work considers a very different data source from code—software tags, and our objective is different, i.e., to produce semantically related tags and their taxonomy.

In other research communities, there are studies on taxonomies of concepts and tags (e.g., [6], [11], [19]). Although having a similar objective, we use different techniques on different data to construct different taxonomies.

Social Media and Software Engineering: Our work is related to but different from many studies on the uses of social media in software development [16]. Treude et al. investigate how tagging is used to improve software development [18]. Dabbish et al. investigate the social coding site GitHub and analyze the impact of its transparency [7]. Pagano and Maalej investigate blogging behaviors among software developers [14]. Gottipati et al. use machine learning tools to categorize posts on software forums [8].

VI. Conclusion and Future Work

In this work, we propose the problem of inferring semantically related software tags that are used to characterize the main features of software products. We propose a new similarity metric for software tags by considering the documents tagged by them. We collect tens of thousands of projects and their tags from Freecode and build a taxonomy of tags based on our metric and the k-medoids clustering algorithm. Our user studies show that our metric and the generated taxonomy capture the actual semantic relations among software tags reasonably well.

In the future, we plan to extend this study by investigating more projects and tags from more project hosting sites with larger scale user studies. We also plan to integrate the taxonomy of software terms into code search engines.

References

[1] "WordNet," http://wordnet.princeton.edu.
[2] S. L. Abebe, S. Haiduc, P. Tonella, and A. Marcus, "Lexicon bad smells in software," in *WCRE*, 2009, pp. 95–99.
[3] ——, "The effect of lexicon bad smells on concept location in source code," in *SCAM*, 2011, pp. 125–134.
[4] J. Beall, "The weaknesses of full-text searching," *Journal of Academic Librarianship*, vol. 34, pp. 438–444, 2008.
[5] L. R. Biggers, B. P. Eddy, N. A. Kraft, and L. H. Etzkorn, "Toward a metrics suite for source code lexicons," in *ICSM*, 2011, pp. 492–495.
[6] B. Cui, J. Yao, G. Cong, and Y. Huang, "Evolutionary taxonomy construction from dynamic tag space," in *Proceedings of the 11th international conference on Web information systems engineering (WISE)*, 2010, pp. 105–119.
[7] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, "Social coding in github: transparency and collaboration in an open software repository," in *CSCW*, 2012.
[8] S. Gottipati, D. Lo, and J. Jiang, "Finding answers in software forums." in *ASE*, 2011.
[9] S. Haiduc and A. Marcus, "On the use of domain terms in source code," in *ICPC*, 2008, pp. 113–122.
[10] J. Han and M. Kamber, *Data Mining Concepts and Techniques*, 2nd ed. Morgan Kaufmann, 2006.
[11] X. Hu, N. Cercone, and J. Han, "Discovery of knowledge associated with concept hierarchy in database," in *Proc. 3rd Int. Conf. for Young Computer Scientists*, Beijing, China, July 1993.
[12] A. Kuhn, "Automatic labeling of software components and their evolution using log-likelihood ratio of word frequencies in source code," in *MSR*, 2009, pp. 175–178.
[13] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[14] D. Pagano and W. Maalej, "How do developers blog? an exploratory study," in *MSR*, 2011.
[15] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *IWPC*, 2002, pp. 271–280.
[16] M.-A. D. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *FoSER*, 2010.
[17] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *ASE*, 2011.
[18] C. Treude and M.-A. D. Storey, "How tagging helps bridge the gap between social and technical aspects in software development," in *ICSE*, 2009.
[19] E. Tsui, W. Wang, C. Cheung, and A. S. Lau, "A concept-relationship acquisition and inference approach for hierarchical taxonomy construction from tags," *Information Processing & Management*, vol. 46, no. 1, pp. 44–57, 2010.
[20] E. M. Voorhees, "Using wordnet to disambiguate word senses for text retrieval," in *SIGIR*, 1993.
[21] X. Wang, D. Lo, J. Jiang, L. Zhang, and H. Mei, "Extracting paraphrases of technical terms from noisy parallel software corpora," in *ACL/AFNLP*, 2009.
[22] J. Yang and L. Tan, "Inferring semantically related words from software context," in *MSR*, 2012.