Fuzzy Set Approach for Automatic Tagging in Evolving Software

Jafar M. Al-Kofahi, Ahmed Tamrawi, Tung Thanh Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen
Electrical and Computer Engineering Department
Iowa State University
Emails: {jafar,atamrawi,tung,hoan,tien}@iastate.edu

***Abstract*—Software tagging has been shown to be an efficient, lightweight social computing mechanism to improve different social and technical aspects of software development. Despite the importance of tags, there exists limited support for *automatic tagging* for software artifacts, especially during the *evolutionary* process of software development. We conducted an empirical study on IBM Jazz's repository and found that there are several missing tags in artifacts and more precise tags are desirable. This paper introduces a novel, accurate, automatic tagging recommendation tool that is able to take into account users' feedbacks on tags, and is very efficient in coping with software evolution. The core technique is an automatic tagging algorithm that is based on fuzzy set theory. Our empirical evaluation on the real-world IBM Jazz project shows the usefulness and accuracy of our approach and tool.**

I. Introduction

Tags are popular in Web communities [1]. They provide an external metadata applied to a Web object such as a Web blog, Web page, a picture, etc. They are used for data searching, data description, identification, bookmarking, or personal markers. For software development, tagging has been shown as a lightweight yet very useful mechanism in helping in developers' communication and bridging the gap between social and technical aspects [2]. It provides a simple way for annotations in which a developer could tag his artifacts so that others could search and find them. With tags, developers could see the broad categories of the tagged artifacts in terms of relevant subjects, purposes, or functions. Tagging is also used for artifact organization. Despite its importance in supporting informal activities in software development, there is little research on the necessary automated supports for software tagging. It was also reported that tagging has not been extensively studied in a software engineering context [2].

Our goal is to identify the key components in an automatic tagging tool for software artifacts, especially an accurate tool that is efficient in the evolutionary process of software development. With that goal in mind, we first wanted to investigate the current usages of tags in a real-world software project. We conducted an empirical study on the IBM's Jazz repository and focused on software tags for work items. Jazz's repository contains the real-world development data from IBM for more than 3 years. A *work item* is a generalized concept of a development task. It contains a summary, a description, a tag, and relevant software artifacts including source code, requirements, test cases, discussions, etc. In the empirical study, our key questions include 1) what are the general purposes and types of tags used in software development, 2) what are the common characteristics of artifacts that share assigned tag(s) from developers, 3) with the current tagging, whether or not the similar work items with the same/similar characteristics/purposes share any tag(s), and 4) with current tagging supports, whether or not the tags are sufficient to distinguish them and to serve those general purposes.

The results of our study showed that developers use tags for three main purposes: 1) categorization of artifacts in different broad concerns, goals, subjects, functions in the system, 2) organization of artifacts in the project, and 3) support for searching of the artifacts. This is consistent with prior findings in [2]. We also found that work items with the same tag(s) often share the same/similar concerns, goals, subjects or functions in the system. It is also generally true (i.e. with a small number of exceptions) that work items that have the same concerns, goals, subjects, or functions are often assigned with the same tags. Interestingly, we also found that many work items that have the similar concerns, goals, subjects, or functions should have additional tags to further and better characterize or describe them with regard to those aspects. Importantly, there is a large percentage of work items that did not have any tag. According to prior research [2], developers recognize the important roles of tags in software development. Therefore, a more accurate and efficient auto-tagging tool that takes into account the contents of work items would improve the current state of software tagging.

With the motivation from our empirical study, we developed an accurate, automatic tag recommendation tool, TagRec, that is able to take into account users' feedbacks on tags, and is very efficient in coping with software evolution. The core of TagRec is an automatic tagging algorithm for textual artifacts that is based on the fuzzy set theory [3]. For each term (i.e. a word within a work item's text that is not a grammatical one or a stopword) collected from the contents of all work items, TagRec defines a fuzzy set. Each work item has a membership value in this set, which signifies the degree of membership of that work item with respect to the fuzzy set defined by a term. The membership value for a work item is in the interval [0,1] with 0 corresponding to no membership in the set defined by the term, and 1 corresponding to full membership. To compute the membership values for all work items with respect to all terms in the corpus, TagRec first builds a correlation matrix

for all meaningful terms (i.e. grammatical terms and stopwords are removed). The correlation value between term k_1 and k_2 in the corpus is defined based on the number of work items in the corpus in which k_1 and k_2 occur together. Then, the membership values are computed based on the principle that a work item w_j belongs to the fuzzy set associated to the term k_i, if many of its own terms in w_j are strongly related to k_i. At last, the membership values for each work item will be sorted and the terms corresponding to the membership values that exceed a chosen threshold will be reported as the tags for that work item.

The key departure points and unique features of our automatic tagging approach from existing ones include 1) its formulation of tagging with the fuzzy set theory that enables the tag recommendation for new work items introduced during software development, 2) its ability to recommend to a work item with tags that do not even appear within its own textual content (due to a mapping scheme between tags not appearing in the texts and the terms appearing in the texts), 3) its ability to improve the future tagging by taking in account the users' feedbacks on the current resulting tags, and 4) its ability to assign tags to items while maintaining the already assigned tags in the project.

We also conducted an empirical evaluation of TagRec in tag recommendation using the IBM's Jazz data. We ran TagRec on Jazz's work items and requested human subjects to evaluate the results from TagRec to see if the resulting tags are good descriptions in terms of both goals, subjects, concerns, or functions of the work items. The same experiment was also carried out to evaluate how well TagRec recommends additional tags to already tagged work items. The results show that TagRec is very time efficient and provides useful recommendation tags with high precision and recall. The third evaluation was conducted in which TagRec was trained in a set of work items and used to recommend for the other set of items in the same project. TagRec also gave high quality results in term of both precision and recall. Our last experiment was carried out in which the feedbacks from human subjects in the first experiment were taken into consideration, and the tool uses that information to re-assign the tags to other work items. Compared to the resulting tags without users' feedbacks, the tagging quality is higher in both precision and recall.

The key contributions of this paper include

1) An empirical study on the characteristics and relationships of work items and their associated tags,
2) TagRec, an efficient and accurate, automatic tag recommendation algorithm that is based on the fuzzy set theory with aforementioned unique features, and
3) A prototype tool and an empirical evaluation on a real-world software project to show the usefulness and accuracy of TagRec.

Section II describes our empirical study on the characteristics and relationships of tags and work items. Section III presents our fuzzy set-based algorithm for automatic tagging. The empirical evaluation is discussed in Section V. Related work is discussed in Section VI. Conclusions appear last.

Data object	Amount
Work items	47,563
Tagged work items	12,889

TABLE I
INFORMATION ON WORK ITEMS EXTRACTED FROM JAZZ REPOSITORY

II. EMPIRICAL STUDY

A. Goals, Hypotheses, and Settings

Tagging has been shown to be a lightweight and useful mechanism for improving the communication and bridging the gap between social and technical aspects in software development [2]. Auto-tagging has also been investigated in the Web research and social computing communities [4], [5]. However, there has been little research on auto-tagging for evolving software [2]. To deeper understand the current state of software tagging in real-world practice and ultimately build an efficient automatic tag recommendation tool, we conducted an empirical study on IBM Jazz's repository, which contains the data for their software development for more than three years. In our study, we collected the quantitative data and extracted all relevant information on the use of tags through accessing the Jazz repository. The data was extracted for the time period from June 2005 to June 2008 (Table I). Before describing the details of our study, some important concepts and definitions are needed.

Definition 1: **A work item** in Jazz is a generalized notion of a development task, which consists of a summary, a description, a tag, and relevant software artifacts.

Definition 2: **Category** is a broad concern for the system, and is usually a functional or non-functional requirement concept such as *performance*, *scalability*, *accessibility*, etc.

Definition 3: **Goal** is an objective to be achieved or fulfilled in a project.

Definition 4: **Subject** is a topic of interest that a work item describes about.

Definition 5: A **tag** is a chosen keyword or term that is associated with or assigned to a work item.

Definition 6: **Similar work items** is a set of work items sharing similar goals, subjects, or categories.

In our study, we took a sufficient period of time to get ourselves familiar with Jazz data and the nature of its work items, and what concerns, goals, and topics of such work items. Then, we aimed to answer the following *research questions* that constitute our empirical study:

1) R1. Do *similar work items* have tags in common?
2) R2. Are there common characteristics among work items sharing the same tags?
3) R3. Do the work items that have some tags in common share also a common category, goal, or subject, but they are not totally *similar work items* in all aspects?
4) R4. Do patronymic tags have the same categories, goals, or subjects of work items, e.g *doc* and *documentation*?
5) R5. Do the tags come from the textual contents?

For brevity, we define the following types of work items.

Data object	Amount
Number of tested clusters	200
Number of work item pairs tested	421
Number of *type 1 set* work items	66

TABLE II
DO SIMILAR WORK ITEMS HAVE TAGS IN COMMON?

Definition 7: **Type 1 set** is a set of *similar work item* pairs that have no tags in common.

Definition 8: **Type 2 set** is a set of work item pairs within the same *category* having tags in common and need more tags to differentiate their *goals/subjects*.

Definition 9: **Type 3 set** is a set of perfectly matched work item pairs within the same *category* and sharing the same *goal*, *subject*, and *tag*.

Prior research [2] on tagging in Jazz's repository via interviewing with 175 developers has concluded that they use tags for the three following general purposes: 1) categorization of broad concerns and goals of the system, 2) organization of artifacts in the system, and 3) searching the relevant artifacts. Moreover, they found that tags are used to describe the goals, subjects, broad concerns, or functions of the work items. The answers for research questions R1-R5 in this study will complement to that prior knowledge because our study investigates the contents of work items and their tags. More importantly, the answers will help us in producing a more precise and useful tag recommendation tool.

B. Activities and Results

For R1, we investigated the similar work items with some tags in common. First of all, we used *WVTool* [6] to analyze the contents of all work items by removing all grammatical terms and stopwords (e.g. "a", "the", "and", etc), stemming the terms, and then producing the significance values for each term with respect to each work item. The significance values are Term frequency - Inverse document frequency (Tf-Idf) [7]. Then, we clustered work items based on their Tf-Idf scores. This step produced 7,823 clusters where each one has at least two work items. This step plays the role of initial filtering and allows us to avoid exhaustive pairwise comparison for all work items. Then, we manually studied 200 clusters and tried to answer the first question for each pair of work items.

To manually verify whether the work items are similar or not, we read their summaries, descriptions, and all relevant information such as related artifacts and discussions. Note that according to Definition 6, similar work items have the similar/same goals, subjects, concerns, or functions. If they are similar and have no tags in common, they were reported as type 1 pairs. Table II shows the results. From the results, there were only 66 work item pairs that share the same goals, subjects, or categories but have no tags in common. Thus, *the answer for R1 is generally true*. However, there are a small number of work item pairs that share similar categories, subjects, or goals but have no shared tags. The implication of this finding is that an automatic tagging tool should take into account the *contents of work items* to provide the tags that reflect better their *subjects* and *goals*, and serve better three purposes in categorization, organization, and searching.

Data object	# tested	Total
Number of work item pairs tested	419	
Number of *type 2 set* work items	356	
Number of *type 3 set* work items	63	
Number of *type 2 set* work items sharing 5 tags	0	3
Number of *type 3 set* work items sharing 5 tags	3	3
Number of *type 2 set* work items sharing 4 tags	31	36
Number of *type 3 set* work items sharing 4 tags	6	36
Number of *type 2 set* work items sharing 3 tags	179	229
Number of *type 3 set* work items sharing 3 tags	50	229
Number of *type 2 set* work items sharing 2 tags	143	147
Number of *type 3 set* work items sharing 2 tags	4	147
Number of *type 2 set* work items sharing 1 tag	3	4
Number of *type 3 set* work items sharing 1 tag	1	4

TABLE III
ARE THERE COMMON CHARACTERISTICS AMONG WORK ITEMS SHARING THE SAME TAGS?

Data object	Amount
Number of work item pairs tested	103
Number of *type 2 set* work items	74
Number of *type 3 set* work items	29

TABLE IV
DO THE WORK ITEM PAIRS HAVING SOME TAGS IN COMMON SHARE ALSO A COMMON CATEGORY, GOAL, OR SUBJECT BUT THEY ARE NOT SIMILAR IN ALL ASPECTS AND MAY NEED ADDITIONAL TAGS?

To answer R2, we clustered the work items based on the shared tags. This results in the pairs of work items sharing: five, four, three, two, or one tag. No pair of work items shares six or more tags. Then, we randomly selected a sample data of 419 work item pairs that range from the pairs sharing five to one tag(s). We manually checked them and found that the common characteristics of work items sharing the same tags fall in the similar/same categories, goals, or subjects. Thus, *the answer for R2 is yes*. We also examined the number of type 2 pairs of work items. Table III summarizes the results. The column $\#tested$ shows the numbers of pairs that we manually verified. We can see that 85% of the work items sharing tags are of *type 2* and the remaining work items are either duplicate ones or having the same tag(s). That is, among those tested work items, 356 (i.e. 85%) of those sharing tags should have more tags to differentiate them in terms of *subjects* or *goals*, despite that they fall into the *same categories*.

To answer for R3, we investigated the work items that share *some* tag(s) but might not share all the assigned tags (Table IV). We randomly picked 103 pairs of such work items. Our manual verification aims to determine whether the work items sharing some tags are the examples of good tagging, and for those that are not, whether more tags are desirable for the work items. Table IV shows the result for this case. From the numbers in Table IV, we can see that 28.1% of the work item pairs are of type 3, which means that these work items were well-tagged, and need no further tag recommendation. It is preferable to have more tags in the other 71.9% of items. This

is consistent with the result in the previous experiment. That is, to serve well for the 3 purposes (categorization, organization, and searching), more tags are desirable even on the work items that were already assigned tags. Thus, *the answer for R3 is yes*. That is, work items having common tags could share some common categories, goals, or subjects. However, they might not necessarily be completely similar in all aspects.

For R4, through our study, we noticed the existence of some patronymic tags, such as: *doc* and *documentation*, *tests* and *testing*, or *decorations* and *decorators*. We aimed to know whether such patronymic tags are just redundant tags that can be merged or they deliberately were used in that way. To conduct our study on patronymic tags, we studied all the work items that were tagged with such tags, and then judged whether such tags can be merged into one tag. For example, in Jazz, *doc* is used to tag work items that are related to the documentation and some enhancements to the GUI comments. *documentation* is also used for work items that talk about documentation in general and some defects in multiple modules. Interestingly, there are some work items that have both *doc* and *documentation*. Also, *tests* and *testing* tags cannot be merged, as *tests* is used to tag work items related to testing and JUnit tests, and other related cases for testing the code. For *testing*, the tag was associated with work items that either are about fixed defects and need to be verified, or are closed work items for being marked as duplicates, etc. On the other hand, *decorations* and *decorators* can be merged. In brief, for the cases we found, all of patronymic tags share the category, and for some cases they even share the subject, thus the answer for R4 is yes.

For R5, we build a simple tool to collect all tags in work items in Jazz. The tags in Jazz can be divided into three parts. More than 40% of the tags appear in the summaries and descriptions of work items, the other 60% do not. Among 60% of tags, there are a few time-constraint tags where such tags are valid for only a specific time range (e.g. *beta2candidate*).

C. Observations

From this empirical study on the real-world data in Jazz's repository, we had learned several insights on tags and tagging:

1) There are several work items that are still missing tags. From Table I, we can see that there are 34,674 (73%) work items that are not tagged. An automatic tag recommendation tool will be very useful in helping the developers in the tagging process. For example, after a work item is created, the tool could analyze relevant artifacts and recommend a set of tags. Developers could choose any of them or enter their own tag(s).
2) Our study confirms the three purposes of tag usages: categorization, organization, and searching. Work items that share tags are often related to the same/similar categories (concerns), goals, or subjects. The work items with the same or similar categories, goals, or subjects generally share the same tags (with few exceptions).
3) There are a large number of work items that should be assigned one or more additional tag(s) to help in further distinguishing them along those aforementioned three aspects, especially on subjects and goals. That information is expressed in the number of type 2 work items. Moreover, goals and subjects of a work item could be learned from its textual contents.
4) During development, software constantly evolves. It is desirable that an auto-tagging tool is able to efficiently work for any new work items without re-running for entire collection of all work items and artifacts. Treude *et al.* [2] also suggested the need of a tag recommendation tool that can recommend tags to handle new work items.

Based on the lessons learned from this study, we propose TagRec, an automatic tag recommendation tool, that can be integrated into the Jazz environment to help recommend tags for 1) already tagged work items by enhancing that tagging, 2) un-tagged, and 3) new incoming work items based on the already tagged work items. The following section discusses our approach to build TagRec using the Fuzzy set theory.

III. Model and Approach

A. Formulation

TagRec is a tagging recommendation tool that automatically assigns tag(s) for any work item. Importantly, it supports the evolutionary process of software work items as well. That is, during software development, it will recommend tags for any new work item while maintaining the existing tags for existing artifacts. It is also useful in the scenario to recommend the tags for current missing-tag work items while keeping the already tagged work items.

Definition 10 (Work Item): A work item is modeled as a sequence of terms (i.e. stemmed words that are not grammatical ones or a stopword) in its title, summary, description, keywords, and related software artifacts.

Definition 11 (Auto-Tagging): Given a set of work items in which some of them could be already associated with some tags and some are not, when a new set of work items is introduced, the auto-tagging tool analyzes the collection of work items and recommends the tags for all work items that do not have tags yet while maintaining the tags for already tagged items and possibly providing additional tags for them.

B. Fuzzy Set Approach

In TagRec, we model the tagging problem based on the fuzzy set theory [8], [9]. Let us describe our model in details. After all stopwords and grammatical terms such as "a", "the", "and", etc are filtered, the remaining terms that appear in all work items or are used as keywords and tags should carry some semantic meaning to work items. All of those terms are collected into a set of terms for consideration in the corpus. Each term defines a fuzzy set and each work item has a degree of membership in this set. The key idea is to associate a membership function for each work item with respect to a particular term. The membership function takes values in the interval [0,1] with 0 corresponding to no membership in the class defined by a term, and 1 corresponding to full membership. Membership values between 0 and 1 indicate

marginal elements of the class. Thus, membership in a fuzzy set is a notion intrinsically gradual, instead of concrete as in conventional logic [9]. That is, each term has a fuzzy boundary and each work item has a membership value indicating that the content of the work item belongs to that boundary.

Definition 12 (Fuzzy Set for a Term): A fuzzy set T in the collection W of all work items is characterized by a membership function $\mu_T : W \rightarrow [0, 1]$, which associates with each element w of W a number $\mu_T(w)$ in the interval [0,1] in which 0 corresponds to no membership and 1 corresponds to full membership [9].

From the perspective of a work item, each of work items w_j has a set of membership values $\mu[i, j]$ between [0,1], signifying the degree of membership it has with respect to each term k_i in the corpus. If sorting all of those membership values $\mu[i, j]$ for all terms k_i in the corpus, one would have the degrees of relevance of the work item w_j with all the terms. In other words, one would have a set of the terms that are most suitable to describe the work item w_j, and the corresponding ranks of the terms according to their degrees of relevance.

The computation of the membership values for all work items with respect to all terms is performed via the computation of term correlation values as follows.

C. Term-Term Correlation

Some information retrieval models consider the terms as independent features (such as in the vector-based model (VSM) [9]). Unlike those models, TagRec examines the relationships among terms via the work items containing the terms. TagRec adopts the concept of keyword connection matrix in [8] to define the term-term correlation values as follows. The term-term correlation matrix is a matrix whose rows and columns are associated to the index terms in the corpus. In this matrix C, a normalized correlation factor $C[i, j]$ between two terms k_i and k_j is defined as

$$C[i, j] = \frac{|D_{i,j}|}{|D_i| + |D_j| - |D_{i,j}|} \tag{1}$$

where $|D_i|$ and $|D_j|$ are the numbers of work items containing the term k_i and k_j respectively, and $|D_{i,j}|$ is the number of work items containing both terms. If a work item is tagged with a term, TagRec considers that the term is contained in that work item.

This term correlation definition is used for the terms appearing in the texts of work items. However, there exist some terms that are used as tags for a work item but do not appear in any other work items in the entire corpus. For example, developers could assign the keyword "performance" to a work item w, however, that term never occurs in any other items. In this case, TagRec defines the term correlation for such terms (referred to as *out-texts*) based on all the terms appearing in w (referred to as *in-texts*). Assume that an out-text tag k_t is assigned to a work item w containing the terms $k_1, k_2, ..., k_p$. Then, the correlation between k_t and k_i and vice versa is defined as

$$\frac{fr(k_i)}{\sum_{x=1}^{p} fr(k_x)} \cdot \frac{|D_{t,i}|}{|D_t| + |D_i| - |D_{t,i}|} \tag{2}$$

The first part of Equation 2 takes into consideration the ratio between the frequency of k_i over the total number of terms in w. The second part is the same as the formula (1). The idea is that k_t will be strongly related to a term k_i if k_i appears many times in the work items that k_t are assigned, however, there are not many work items containing k_i in the entire collection. For a term that does not occur in w, the correlation value between k_t and that term is defined as zero. Of course, all stopwords and grammatical terms are filtered before the term frequency counting. This mapping between out-texts to in-texts enables TagRec to take into account the existing tags (as out-texts) in some work items. Then, when some other work item w' that is semantically related to w, TagRec could recommend the out-text tag k_t to w' even though the work item w' does not contain the tag k_t in its content at all.

D. Membership Values

Definition 13 (Membership Value): A work item w_j has a degree of membership $\mu[i, j]$ with respect to the fuzzy set corresponding to term k_i. The value $\mu[i, j]$ is computed as in [9]:

$$\mu[i, j] = 1 - \prod_{k_l \in w_j} (1 - C[i, l]) \tag{3}$$

The idea is that a work item w_j belongs to the fuzzy set associated to the term k_i, if its own terms are strongly related to k_i. If there exists at least one term k_l occurring within the work item w_j which is strongly relevant to k_i (i.e., $C[i, l] \approx 1$), then $\mu[i, j] \approx 1$, and the term k_i is a good fuzzy index for the work item w_j. If all terms in w_j are irrelevant to k_i, the term k_i is not a good fuzzy index for w_j (i.e. $\mu[i, j] \approx 0$).

E. Additional Ranking Scheme

For each work item w_j, the membership function $\mu[i, j]$ of a term k_i shows us how well the term k_i reflects the content of w_j. However, in formula (3), if there exists at least one term k_l in w_j that is strongly related to k_i, then the membership value is 100%. To distinguish between work items that have more than one such terms k_ls, TagRec introduces an additional ranking scheme. If a work item w_1 that has m terms that are strongly relevant to k_i and another work item w_2 with n terms ($m > n$) strongly relevant to k_i, then the work item w_1 is considered to be more relevant to the term k_i than w_2. Thus, with this ranking scheme, for a given tag k_i, TagRec could rank and return the list of relevant work items.

F. Tag Recommendation

For each work item w_j, all the membership values corresponding to all the terms will be computed as in the formula (3). The terms that give the higher membership values for w_j than a chosen threshold will be returned to developers as the recommended tags.

With the mapping scheme between out-texts and in-texts, TagRec is able to recommend the tag(s) that do not need to occur within the texts of that work item.

For the work items that were already tagged, those tags will be maintained the same for such items because the

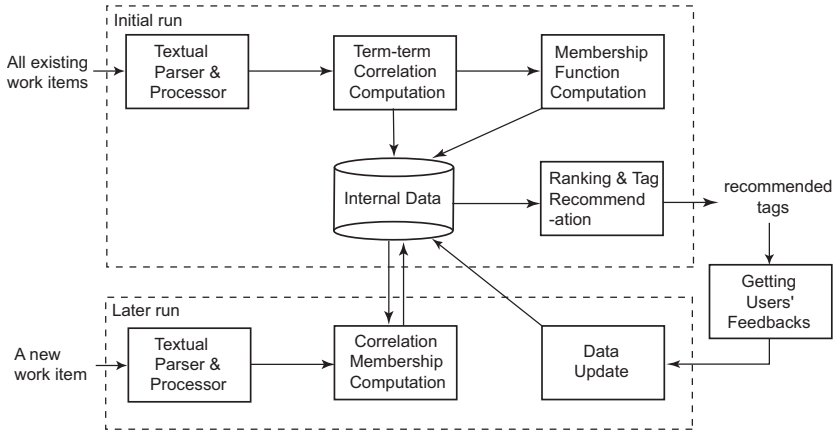

Fig. 1. TagRec Architecture

membership values corresponding to those existing tags will be among the highest. Such work items have at least one term (the tag itself) that is strongly related to the tag. Moreover, additional tags would also be recommended to such existing work items if their corresponding membership values exceed a chosen threshold. This is very useful as our empirical study (Section 2) has shown that there are several related work items requiring additional tagging to be more precisely described. This is a big advantage of our fuzzy set approach in terms of efficiency to cope with software evolution, in comparison with other feature extraction approaches such as Tf-Idf [7], in which when a new document is introduced, all significant values must be recomputed.

For a newly created work item, the membership values with respect to all terms are computed. The new tag(s) will be recommended in the same manner for a newly introduced work item. In other words, our tagging tool is able to work well in the *evolutionary process* of software development with new work items.

G. Users' Feedbacks Integration

When a new work item is created, TagRec recommends to the developer a set of tags. (S)he is able to accept or refuse any tag and provide new tags for the work item. Let us denote the sets of accepted, rejected, and new tags for a work item w_j by T_a, T_r, and T_{new}, respectively.

For the tags in T_a, their membership values with respect to all work items will be kept the same. For the tags in T_r (i.e. being rejected), the membership value μ with respect to the work item w_j will be reduced in half.

For the tags that are newly assigned by users (i.e. in T_{new}), the membership μ with respect to w_j will be assigned as 100%. Then, TagRec will re-compute the membership values for each tag in T_{new} with respect to all other work items in the collection. During the computation, w_j is assumed to *contain* all the new terms in T_{new}. If one of such values exceeds a chosen threshold, the corresponding work item will be associated with the corresponding new tag. That is, that new tag will be recommended for that work item.

IV. Tool Development

We have implemented the aforementioned tagging algorithm with all of those features into a prototype tool, called TagRec. Figure 1 displays an overview of the architecture of TagRec. The figure shows three main parts of TagRec: 1) initial run, 2) later run, and 3) users' feedbacks and updating.

For the initial execution on the large set of work items, the parser module performs all parsing, stemming, and textual analysis tasks. Then, TagRec will compute the term-term correlation matrix and the membership values for each work item with respect to each meaningful term in the corpus. Some important information will be stored as TagRec's internal data. For each work item, TagRec sorts the membership values, produces and presents to developers the recommended tag(s). When a new work item is introduced, TagRec will parse its content. Based on the internal data on existing tags, it partially re-computes the term-term correlation and membership values. Then, TagRec performs ranking and producing tag recommendation as in the initial run. When the tags are recommended to the developers, they are allowed to provide the feedbacks by accepting or rejecting the recommended ones, or adding their own terms. TagRec will update the internal data, adjust the ranking of terms, and produce the new tags if possible.

V. Empirical Evaluation

We have conducted an empirical evaluation on TagRec. Our goal is to evaluate 1) how well TagRec could recommend the tags for work items that have not assigned any tags (i.e. un-tagged work items), 2) how well it could recommend additional tags for the work items that already had some tags, 3) how well it recommends tags for newly introduced work items, 4) how well users' feedbacks affect the results, and 5) how time-efficient our recommendation tool is. We continue to use the IBM Jazz's work items for our evaluation. All experiments were carried out on a Windows Vista, Intel Core 2 Duo 2.10Ghz, 4GB RAM desktop.

A. Experiment 1

Our goal in this first experiment is to answer the first question: how well TagRec could recommend the tags for

work items that have not assigned any tags. The quality of tag recommendation is based on two metrics: precision and recall. **Precision** is defined as the ratio between the number of correctly recommended tags over the total number of recommended tags. **Recall** is defined as the ratio between the number of correctly recommended tags over the total number of needed tags. For the ground truth, we relied on the human subjects' judgement. For human subjects, we selected one MSc. and one Ph.D. student in Software Engineering at Iowa State University with the average of 8-9 years of experience in programming and with 10-12 months of experience in Jazz's data and environment.

Firstly, we executed TagRec on all 47,563 work items. It computed the term-term correlation matrix, membership values, and recommend tags for all items. It is impossible to check the tags for all work items. Thus, we randomly selected 200 work items and their assigned tags by TagRec. Those 200 work items were selected such that they were *not* assigned tags before in Jazz's data because in this first experiment, we wanted to evaluate how well TagRec recommended the tags for un-tagged work items. For each work item, TagRec was configured to output up to 5 tags. The reason we chose an upper limit of 5 tags for each work item because many of work items in Jazz's data have 3-5 tags. Each subject was asked to verify the recommended tags for 100 work items whether each tag is correct or not. Subjects were also requested to provide additional tags if they felt that the suggested tags were insufficient. Subjects were asked to verify and provide additional tags (if needed) for all 200 items. They also did cross validation and verification of each others' results. We collected the results for all 200 ones. These provided tags from human subjects and the verified tags were used as the ground truth to compute the recall of tag recommendation. The total number of correctly recommended tags is determined based on the human verification on the output tags from TagRec.

More specifically, let us assume that we have the work items w_1 to w_N in our testing set. For each w_i, let us denote the number of resulting tags by r_i, and the number of correct tags among the resulting tags r_i identified by human subjects by c_i. Let us use s_i to denote the number of additionally suggested tags by subjects. Then, precision is computed as

$$\frac{\sum_{i=1}^{N} c_i}{\sum_{i=1}^{N} r_i} \tag{4}$$

and recall is computed as

$$\frac{\sum_{i=1}^{N} c_i}{\sum_{i=1}^{N} (c_i + s_i)} \tag{5}$$

The results of this first experiment is shown in Table V. We could see that TagRec could recommend the tags for un-tagged work items with very good recall (approximately 70%) and reasonably good precision levels (51%). In tag recommendation, high recall is more preferable because the majority of needed tags will be brought into developers' attention. Developers will not need to scan through and understand the contents of many work items in the project to determine the possible tags. On the other hand, the incorrect tags among suggested ones could easily be discarded by developers. Thus, this result shows the usefulness of TagRec for tag recommendation for un-tagged work items.

Data object	Amount
# tested work items (N)	200
# recommended tags by TagRec	646
# recommended tags by subjects	145
# correct tags	328
Recall	69.3%
Precision	50.8%

TABLE V
TAG RECOMMENDATION FOR UN-TAGGED WORK ITEMS

Data object	Amount
# tested work items (N)	200
# additional recommended tags by TagRec	668
# recommended tags by subjects	109
# correct tags	305
Recall	73.7%
Precision	45.7%

TABLE VI
ADDITIONAL TAG RECOMMENDATION FOR TAGGED WORK ITEMS

B. Experiment 2

Our goal in this experiment is to answer the second evaluation question: how well TagRec could recommend additional tags for tagged work items. The data set and the settings of this experiment are similar to experiment 1. We also executed TagRec on all 47,563 work items. However, for manual checking, we randomly selected 200 work items that did have tags originally in the Jazz's data, and then requested the human subjects to examine the additional tags recommended by our tool. Subjects were also requested to provide additional tags if they found that the suggested tags were not sufficient. Similar to the previous experiment, the results for all 200 work items were collected. Subjects' inputs were used as the ground truth for computing precision and recall.

Let us use a_i to denote the additional tags that TagRec recommends for work item w_i. Among a_i tags, assume that the subjects selected c'_i as the correct ones and provided s_i additional tags by themselves. Then, precision is computed as

$$\frac{\sum_{i=1}^{N} c'_i}{\sum_{i=1}^{N} a_i} \tag{6}$$

and recall is computed as

$$\frac{\sum_{i=1}^{N} c'_i}{\sum_{i=1}^{N} (c'_i + s_i)} \tag{7}$$

The result of this experiment is shown in Table VI. On average, TagRec recommends 3 additional tags for one work item. In this experiment, it recommends *additional tags* for already tagged work items with even higher recall and slightly lower precision. Because TagRec's fuzzy set approach considers that a work item contains its associated tag(s), the tags that

Data object	Amount
# tested work items (N)	100
# recommended tags by TagRec	368
# recommended tags by subjects	36
# correct tags	145
Recall	80.1%
Precision	39.4%

TABLE VII
TAG RECOMMENDATION FOR NEW WORK ITEMS

exist already for those items are kept the same. We manually checked the additionally recommended tags. The correct ones mostly reflect the goals and subjects of the work items.

C. Experiment 3

In this experiment, we want to evaluate how well TagRec could recommend tags for new work items. Since it is impractical for us to add new work items into existing Jazz's data, we chose to evaluate this aspect via a different way. We sorted all tagged work items in Jazz with respect to their creation dates from the oldest to the newest. The newest 100 work items were used for testing and the older work items were used for training TagRec. The contents of those 100 work items were also analyzed during the recommendation process of TagRec. The recommended tags for those 100 work items were verified by a human subject. Precision and recall was computed in the same way as in the previous experiments. The result of this experiment is shown in Table VII. In this case, TagRec recommends for the 100 newest work items with a higher recall and lower precision. We manually examined the incorrect tags and work items. We found that many of them have brief and short textual contents.

D. Experiment 4

Our goal in this experiment is to answer the fourth evaluation question: how well TagRec's feedback approach affects tag recommendation results. For this experiment, we reused the tagging result for 100 work items from experiment 1, and considered the tags from the human subject 1 as the feedback to the result. We took into account the feedback on the correctly recommended tags by TagRec and the human-suggested tags by the human subject 1, and executed TagRec with that feedback to update its internal data. Then, we used TagRec to recommend the tags on the totally different 100 work items checked by the human subject 2 to see if the tag recommendation result for such items improves. Note that we did not take the feedback from the human subject 2 into consideration. In this experiment, we are interested in the affect of users' feedbacks on the recommendation, so for manual checking, 1) we checked for any improvements in the quality of the recommended tags, 2) we checked the affect of users' feedbacks on the recall and precision which will be calculated in the same way as in experiment 1, 3) we checked for any newly added tags that did not exist before the feedback and was recommended now, and 4) we checked any changes in ranking of the tags for each work item by comparing them to the results of the run in experiment 1.

Data object	# No Feedback	With Feedback
# Recommended tags by TagRec	333	340
# Recommended tags by subjects	65	61
# Correct tags	154	163
Recall	70%	73%
Precision	46%	48%

TABLE VIII
USERS' FEEDBACKS IMPROVE TAGGING RESULT

Work Items	Terms	Rec_Tags	PPTime	RecTime
47,563	52,708	166,470	54 secs	45 mins

TABLE IX
TIME EFFICIENCY

Table VIII shows the results for the first two parts of experiment 4. The table compares the results from subject 2 for experiment 1 without the feedback (shown under the No Feedback column), and the same set but with the feedback under the With Feedback column. From Table VIII, we could see the improvement in the number of correct tags recommended by TagRec, and the reduction in the number of additional tags by human verification. Both indicates that using users' feedbacks in TagRec improves the quality of the recommended tags, reduces the need for manually added tags from the users, and increases both recall and precision.

To check for goal 3, we asked the human subject 2 to verify the rank of every recommended tag for every work item in his set and report any increasing/decreasing in the correctly recommended tags. Also, we provided him with a list of all new tags that were added from the feedback set from the human subject 1, and asked him to report if any tags from the feedback are used in tagging for his data set (i.e. the second 100 work items). The result is that there were a total of 14 new tags provided as the feedback from subject 1. Seven of them were applicable and used as tags for 13 times in the second 100 work items. 10 out of those were *correct* recommendations and 3 were not. For goal 4, we requested subject 2 to check how many correct tags we gained and how many we lost (i.e. it made the tag's rank low enough to be ignored by TagRec). The result is that TagRec was able to increase the membership scores for 3 different tags for 3 work items. Those 3 tags were ignored in the run without the feedback. Importantly, TagRec did not lose any correct tag.

E. Time Efficiency

For time efficiency, we evaluated the running time for both the initial execution on a large number of work items, and the later run for a single work item. For the initial run, we measured the running time as TagRec was executed in all 47,563 work items in Jazz's repository. Table IX shows the result for the initial run. The columns PPTime and RecTime display the time for preprocessing (including textual parsing), and the time for tag recommending (including term-term correlation computation, membership computation, and ranking). The recommendation time was about 45 minutes for a very large amount of work items in about 3 years of development

at IBM. For the second case, we repeated the experiment 3 on the newest 100 work items, however, TagRec was executed for each of those 100 items at a time. On average, it took 3 seconds to recommend for one work item. These results show that TagRec is scalable, efficient, and well-suited for interactive use for tag recommendation as a new work item is introduced.

F. Threats to Validity

Our human subjects are not the real authors of the work items in Jazz. Therefore, the judgements might not be the same. However, because the human subjects have experience in both programming and Jazz environment, and they were requested to examine all relevant documents to a work item, their verifications are trustworthy. Moreover, our result is currently only on the Jazz project.

VI. Related Work

Tagging has been used to help in the social and technical aspects in software development. According to Treude and Storey [2], a tag is "a freely chosen keyword or term that is associated with or assigned to a piece of information. In the context of software development, tags are used to describe resources such as source files or test cases in order to support the process of finding these resources". The resources here could be regarded in a general way including any software artifacts as well as "work items" as defined in IBM's Jazz (i.e. all artifacts relevant to a particular task in software development). In their study [2], the authors have shown the important roles of tags in many informal processes in software development and that the tagging mechanism was eagerly adopted by the team. Their inspection of the Jazz project's repository shows that a large number of work items are missing tags and that the suggested tags from the Jazz tool are not favored by developers.

Despite the popularity of automatic tagging tools in Web environment and social computing technologies, there has not been many such tools in software development. TagSEA (Tags for Software Engineering Actitivies) supports social tagging and communication among developers via tags and user-defined navigational structures [10], [11]. Other types of tools supporting for tagging in source code include GNU Global [12], social bookmarking in Code Snipets [13], Byte-Mycode [14]. Unlike our tool, these tools focus on source code and have limited supports for other types of software artifacts.

Poshyvanyk *et al.* [15] use Latent Semantic Indexing (LSI) and formal concept analysis (FCA) to map the concepts expressed in users' queries into code. Those techniques are used to map the concepts expressed in queries written by the programmers to relevant parts of the source code, presented as a ranked list of search results. Poshyvanyk *et al.* combine LSI with scenario-based probabilistic ranking for feature identification [16]. Their results on Mozilla show that the combined technique improves feature identification significantly with respect to each technique used independently. They also use LSI in coupling concepts in OO systems [17]. Liu *et al.* [18] propose an approach for feature location via combining information from two different sources: an execution trace, and the comments and identifiers from the source code.

Hindle *et al.* [19] use Latent Dirichlet Allocation (LDA) and LSI to extract a set of independent topics in developers' discussions from a corpus of commit-log comments. Gay *et al.* [20] propose the use of relevant feedbacks to improve concept location with various information retrieval (IR) techniques. Liu *et al.* [21] introduce a measurement for class cohesion via mixtures of latent topics using LDA. Information about relevant documents could be achieved with traceability link recovery (TLR) tools [22]. Antoniol *et al.* investigate two IR methods based on both vector space and probabilistic models [23]. ADAMS Re-Trace is a LSI-based traceability link recovery tool for different types of artifacts in ADAMS, an artifact management system [24], that provides searching and tracing supports. De Lucia *et al.* [25] proposed an TLR process using users' feedbacks, aiming at gradually identifying a threshold that achieves a good balance between retrieved correct links and false positives. COCONUT [26] is able to show the similarity level between high-level artifacts and source code during development using LSI. It can guide programmers to write more understandable code by suggesting the meaningful identifiers from high-level artifacts.

Despite their successes, there are no existing approaches that apply fuzzy sets into automatic tag recommendation for software artifacts. To the best of our knowledge, our approach is the first to apply fuzzy set theory in automatic tagging for software artifacts. Moreover, in comparison with our fuzzy set-based tagging approach, those existing approaches such as LDA, LSI, FCA are more computationally heavy-weight. Those approaches are not quite efficient in dealing with software evolution with new artifacts. They must re-perform the entire process. Furthermore, the mapping between out-texts and in-texts allows TagRec to recommend for a work item the tags that do not occur in its content at all. This improves the tagging quality in TagRec. Finally, users' feedbacks could be nicely integrated into TagRec to improve future tagging.

Examplar [27] aims to find relevant applications via a search engine that is based on concept location and program analysis. Users enter a natural language query that contains high-level concepts. A keyword from the query is matched against the descriptions of different documents that describe API calls of widely used software packages. Wursch *et al.* [28] propose a framework with the use a guided-input natural language to query for information about a software system. It uses ontology and knowledge processing technologies from Semantic Web for query and retrieval. Fritz and Murphy [29] introduce an information fragment model that is based on graphs of information fragments, and allows the composition of different kinds of information to help developers to easily choose how to display the composed information.

Automatic tagging is also popular in Web and social computing areas. In those areas, there exist literature surveys on different tagging systems and classifications of user tags [30], [31]. Song *et al.* [32] use spectral recursive embedding clus-

tering and a two-way Poisson mixture model for real-time tagging of Web documents. TagAssist [1] is an automatic tagging tool for new Web blog posts by utilizing existing tagged posts. It performs lossless compression over existing tag data. P-TAG [33], a method which automatically generates personalized tags for Web pages, produces keywords relevant both to textual content and to the data residing on the surfer's Desktop using Semantic Web technology. Brook *et al.* [4] propose auto-tagging with hierarchical clustering. They show that clustering algorithms can be used to reconstruct a topical hierarchy among tags.

VII. Conclusions

Software tagging has been shown to be an efficient, lightweight social computing mechanism to improve different social and technical aspects of software development. Despite the importance of tags, there exists limited support for *automatic tagging* for software artifacts. We conducted an empirical study on IBM Jazz's data and found that there are several missing tags in artifacts and more precise tags are desired. This paper introduces an accurate, automatic tag recommendation tool that is able to take into account users' feedbacks on resulting tags, and is very efficient in coping with software evolution. The core technique is a fuzzy set-based automatic tagging algorithm. The unique features of our algorithm from existing ones include its ability 1) to efficiently handle the tag recommendation for new work items introduced during software development, 2) to recommend to a work item with tags that do not even occur within its own textual content, 3) to improve the future tagging by taking into account the users' feedbacks on the current resulting tags, and 4) to assign tags to items while maintaining the already assigned tags.

Our empirical evaluation on the real-world IBM Jazz project shows that TagRec is time efficient and well-suitable for interactive uses in daily development. Importantly, it is very accurate with high recall (76%) and precision (50%). Our future work will investigate the use of program analysis to enrich the semantic relations between source-code work items.

Acknowledgment

We would like to thank IBM corporation for awarding us the IBM's Jazz repository that was used in this research.

References

[1] S. C. Sood and K. Hammond, "TagAssist: Automatic Tag Suggestion for Blog Posts," in *International Conference on Weblogs and Social*, 2007.

[2] C. Treude and M.-A. Storey, "How tagging helps bridge the gap between social and technical aspects in software development," in *ICSE '09: 31st Int. Conference on Software Engineering*, pages 12–22. IEEE CS, 2009.

[3] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall, 1995.

[4] C. H. Brooks and N. Montanez, "Improved annotation of the blogosphere via autotagging and hierarchical clustering," in *WWW '06: 15th international conference on World Wide Web*, pp. 625–632. ACM, 2006.

[5] G. Begelman, "Automated tag clustering: Improving search and exploration in the tag space," in *In Proc. of the Collaborative Web Tagging Workshop at WWW 2006*. ACM, 2006.

[6] "WVTool," http://wvtool.sourceforge.net/.

[7] G. Salton and C. Yang, "On the specification of term values in automatic indexing," *Journal of Documentation*, vol. 29, no. 4, pp. 351–372, 1973.

[8] Y. Ogawa, T. Morita, and K. Kobayashi, "A fuzzy document retrieval system using the keyword connection matrix and a learning method," *Fuzzy Sets and Systems*, vol. 39, pp. 163–179, 1991.

[9] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison Wesley, 1999.

[10] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby, "Shared waypoints and social tagging to support collaboration in software development," in *CSCW '06: Proceedings of the 20th conference on Computer Supported Cooperative Work*, pages 195–198. ACM, 2006.

[11] M. A. Storey, L. T. Cheng, J. Singer, M. Muller, D. Myers, and J. Ryall, "How programmers can turn comments into way points for code navigation," in *ICSM'07: International Conference on Software Maintenance*. IEEE CS, 2007.

[12] "GNU global system," www.gnu.org/software/global/.

[13] "Code snippets," http://snippets.dzone.com/.

[14] "byteMyCode," http://bytemycode.com/.

[15] D. Poshyvanyk and A. Marcus, "Combining formal concept analysis with information retrieval for concept location in source code," in *ICPC'07: Int. Conference on Program Comprehension*. IEEE CS, 2007.

[16] D. Poshyvanyk, A. Marcus, V. Rajlich, Y.-G. Gueheneuc, and G. Antoniol, "Combining probabilistic ranking and latent semantic indexing for feature identification," in *ICPC'06: International Conference on Program Comprehension*, pages 137–148. IEEE CS, 2006.

[17] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *ICSM'06: International Conference on Software Maintenance*, pages 469–478. IEEE CS, 2006.

[18] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *ASE '07: 22nd international conference on Automated software engineering*, pages 234–243. ACM, 2007.

[19] A. Hindle, M. W. Godfrey, and R. C. Holt, "What is hot and what is not: Windowed developer topic analysis," in *ICSM'09: International Conference on Software Maintenance*. IEEE CS, 2009.

[20] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in IR-based concept location," in *ICSM'09: International Conference on Software Maintenance*. IEEE CS, 2009.

[21] Y. Liu, D. Poshyvanyk, R. Ferenc, T. Gyimothy, and N. Chrisochoides, "Modeling class cohesion as mixtures of latent topics," in *ICSM'09: International Conference on Software Maintenance*. IEEE CS, 2009.

[22] G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," *Handbook of Soft. Eng. and Knowledge Eng.*, vol. 3, 2005.

[23] G. Antoniol, G. Canfora, G. Casazza, A. de Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[24] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "ADAMS Re-Trace: A Traceability Recovery Tool," in *CSMR'05: European Conference on Software Maintenance & Reengineering*, pages 32–41. IEEE CS, 2005.

[25] A. D. Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *ICSM'06: Int. Conference on Software Maintenance*, pages 299–309. IEEE, 2006.

[26] A. D. Lucia, M. D. Penta, R. Oliveto, and F. Zurolo, "COCONUT: COde COmprehension Nurturant Using Traceability," in *ICSM'06: International Conference on Software Maintenance*. IEEE CS, 2006.

[27] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A Search Engine For Finding Highly Relevant Applications," in *ICSE '10: 32nd International Conference on Software Engineering*. IEEE CS, 2010.

[28] M. Wursch, G. Ghezzi, G. Reif, and H. C. Gall, "Supporting Developers with Natural Language Queries," in *ICSE '10: 32nd International Conference on Software Engineering*. IEEE CS, 2010.

[29] T. Fritz and G. C. Murphy, "Using Information Fragments to Answer the Questions Developers Ask," in *ICSE '10: 32nd International Conference on Software Engineering*. IEEE CS, 2010.

[30] S. A. Golder and B. A. Huberman, "Usage patterns of collaborative tagging systems," *J. Inf. Sci.*, vol. 32, no. 2, pp. 198–208, 2006.

[31] T. Hammond, T. Hannay, B. Lund, and J. Scott, "Social bookmarking tools (I): A general review," *D-Lib*, vol. 11, no. 4, 2005.

[32] Y. Song, Z. Zhuang, H. Li, Q. Zhao, J. Li, W.-C. Lee, and C. L. Giles, "Real-time automatic tag recommendation," in *SIGIR'08: ACM conference on Research and development in information retrieval*, pages 515–522. ACM, 2008.

[33] P.-A. Chirita, S. Costache, W. Nejdl, and S. Handschuh, "P-TAG: large scale automatic generation of personalized annotation tags for the web," in *WWW '07: 16th Int. Conference on World Wide Web*. ACM, 2007.