11-2011

Finding Relevant Answers in Software Forums

Swapna GOTTOPATI
Singapore Management University, SWAPNAG@smu.edu.sg

David LO
Singapore Management University, davidlo@smu.edu.sg

Jing JIANG
Singapore Management University, jingjiang@smu.edu.sg

Finding Relevant Answers in Software Forums

Swapna Gottipati, David Lo, and Jing Jiang
School of Information Systems
Singapore Management University
{swapnag.2010,davidlo,jingjiang}@smu.edu.sg

***Abstract*—Online software forums provide a huge amount of valuable content. Developers and users often ask questions and receive answers from such forums. The availability of a vast amount of thread discussions in forums provides ample opportunities for knowledge acquisition and summarization. For a given search query, current search engines use traditional information retrieval approach to extract webpages containing relevant keywords. However, in software forums, often there are many threads containing similar keywords where each thread could contain a lot of posts as many as 1,000 or more. Manually finding relevant answers from these long threads is a painstaking task to the users. Finding relevant answers is particularly hard in software forums as: complexities of software systems cause a huge variety of issues often expressed in similar technical jargons, and software forum users are often expert internet users who often posts answers in multiple venues creating many duplicate posts, often without satisfying answers, in the world wide web.**

To address this problem, this paper provides a semantic search engine framework to process software threads and recover relevant answers according to user queries. Different from standard information retrieval engine, our framework infer semantic tags of posts in the software forum threads and utilize these tags to recover relevant answer posts. In our case study, we analyze 6,068 posts from three software forums. In terms of accuracy of our inferred tags, we could achieve on average an overall precision, recall and F-measure of 67%, 71%, and 69% respectively. To empirically study the benefit of our overall framework, we also conduct a user-assisted study which shows that as compared to a standard information retrieval approach, our proposed framework could increase mean average precision from 17% to 71% in retrieving relevant answers to various queries and achieve a Normalized Discounted Cumulative Gain (nDCG) @1 score of 91.2% and nDCG@2 score of 71.6%.

I. Introduction

During software development and maintenance activities, users and developers often face issues and questions to be solved. Addressing these questions fast would make maintenance activities cheaper to perform. Fortunately, often questions faced by one developer or user have been faced by many others before. These questions along with the associated conversations (i.e., answers, contexts, etc), are often stored in the many online software forums.

An online forum is a web application for holding discussions. Users post questions, usually related to some specific problems, and rely on others to provide potential answers. Within a forum, there are many threads. And in each thread, there are many posts. Software forums contain a wealth of knowledge related to discussions and solutions to various problems and needs posed by various developers and users. Therefore, mining such content is desirable and valuable.

We investigated over 10 software forums and found that all of them contain question-answer knowledge. The threads contain posts ranging anywhere from 2 to 10,000. It is a painstaking process for users to manually search through many posts in various threads. This is true, especially, for long threads that contains hundreds or even thousands of posts. When the user scans through the posts, he/she often lands up finding a variety of replies to various questions; some of these might not be of interest to him/her. Even after performing an exhaustive search it may turn out that either there are no replies to the question or correct answer has been not be provided.

Forum thread usually consists of an initiating post and a number of reply posts. The initiating post usually has several questions and the reply posts usually contain answers to the questions and perhaps new questions or clarifying information or some kind of feedback. The threads at times grow long either because the issue is hard to solve or new independent initiating questions are posted in the same thread. The new initiating questions will have more replies and the process continues. We refer to a sequence of posts related to a particular initiating question post as a *question-answer conversation*. Within a thread, there could be multiple independent question-answer conversations each starting with a different initiating question post.

Another interesting observation is that the majority of software forum threads contain questions and answers rather than junks or irrelevant contents. It is different from social networking site like Twitter [30]. In a thread, the first post is likely a question and the following posts may contain related information like clarifying posts or potential solutions and feedbacks. Thus, software discussion boards contain rich information similar to organized QA services like Yahoo! Answers that are designed specifically for question answering purpose. Unfortunately, this wealth of information is often hidden in many threads of posts, some of which are very long.

To automatically search for relevant answers, typically developers search via a standard search engine (e.g. Google) or specialized search engines in software forums. The former approach will return many webpages many of which are often not relevant to answering questions. The later approach would return specialized pages from software forums that often contain answers to various question. However, even in the second approach returned answers could be numerous. Consider searching answers for the following question: " How to get values from arraylist?". As shown in the Figure 1,

978-1-4577-1639-3/11/$26.00 © ASE 2011, Lawrence, KS, USA

searching this query in Oracle forum, would return 287 threads with some threads as large as 30 posts. It is the user's job to filter the answers manually across all the threads which is very time consuming.

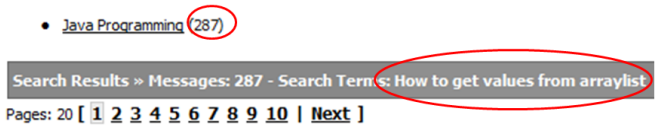

Fig. 1. 287 search results from Oracle forum for query: " How to get values from arraylist?"

To leverage the wealth of information in software forums, we propose a framework to find relevant answers from software forums. Our framework consists of two main steps. First, we propose an engine that automatically classifies and tags posts in software forums as: answers, clarifying answers, clarifying questions, feedbacks - both positive and negative, and junk e.g., "today I am happy". Users could then focus on reading only the questions/answers including those buried deep inside long threads rather than the entire posts. Some questions with correct answers (based on the positive or negative feedbacks) could also be identified. Second, we build a semantic search engine framework that uses the inferred semantic tags to recover the relevant answers from the threads.

We collected a dataset of 4020, 680, and 1368 posts from Oracle, SoftwareTipsandTricks and DZone software forums respectively. Using this dataset, we test our tag inference engine and show that we could infer tags with 67% precision, 71% recall, and 69% F-measure. To evaluate our overall search engine framework, based on the same dataset, we conduct experiments where users are tasked to label returned answers to 17 technical software-related queries expressed in natural language returned by a standard information retrieval toolkit [19] and our proposed framework. We show that we could increase the mean average precision from 17% to 71%, after the tags are utilized. We show that we could achieve nDCG@1 of 91.2% and nDCG@2 of 71.6% on these queries, with automatically generated inferred tags.

The contributions of this work are as follows:

1. We propose an engine that infers a comprehensive set of tags of posts in software forums: questions, answers, clarifying questions, clarifying answers, positive feedback, negative feedback, and junk.
2. To the best of our knowledge, we are the first to propose a semantic search engine to find relevant answers to queries by leveraging automatically inferred tags.
3. We have experimented our solution on 3 real software forums analyzing a total of 6068 posts which shows our framework's reasonable overall accuracy in inferring tags.
4. We have run a user-assisted study to evaluate the quality of our proposed framework in returning relevant answers from software forums with encouraging result.

This paper is organized as follows. Section II describes the related work. Section III presents the details of the forum data extraction process and some of the properties of the forum data that we consider in this study. Section IV describes tag inference engine. Section V presents our semantic search engine framework built upon the tag inference engine. Section VI elaborates our experimental settings, evaluation approaches, results and analysis. Section VII discusses some interesting issues and future work. Finally, we conclude in Section VIII.

II. Related Work

Recently there has been an active interest to mine or extract information from the mass of available software data. Some work propose extraction of information from code [18], [17], [28], [22], execution traces [33], [25], [5], software repositories like SVN or CVS [36], [23], etc. In this work, we are also extracting information from the mass of available software data. Different from many of the above work, we are interested in the inference of tags of posts in software forums and the utilization of those tags to effectively retrieve relevant answers.

There are a number of work on extracting software knowledge from the web or via natural language processing techniques [28], [32], [35], [7], [3]. Similar to us, Thummalapenta and Xie also extract information from the web [28]. However, different from their work, we process textual information in software forums rather than code from Google code. Wang et al. use natural language and execution trace information in bug reports to detect duplicate bug reports [32]. Similar to Wang et al. we also analyze textual information. However, we focus on the retrieval of relevant answers in software forums rather than detecting duplicate bug reports. Zhong et al. mine for software specifications from textual software documentations [35]. Different from their work, we infer tags from software forums rather than specifications from software documentations.

There have been a number of recent studies that analyze logged communication among users and developers to aid software engineering activities. One of the early work is the work by Bird *et al.* in [2] that extracts a social network from developers communication via email. They find that the level of email activity strongly correlates with the level of activity in the source code. Rigby and Hassan perform a psychometric text analysis on OSS mailing list [24]. They find interesting patterns that relate a particular sentiment with a particular release. Wolf et al. investigate the use of social network and developer communication information to predict for failures [34]. Recently, Storey propose the use of feeds to help reverse engineer in leveraging resources from the community (i.e., crowdsourcing) [27]. Dit and Marcus investigate the readability of defect reports [7]. Breu et al. suggest ways how a bug tracking system could be improved with user participation [3]. In this study, we add to the variety of work that analyzes logged communication in particular software forums to aid both developers and users of software products.

Ibrahim et al. analyze forums to find discussion threads that developers could contribute in; in effect answering the question: should I contribute to this discussion? [13]. In this study, we extend their work in analyzing software forums by

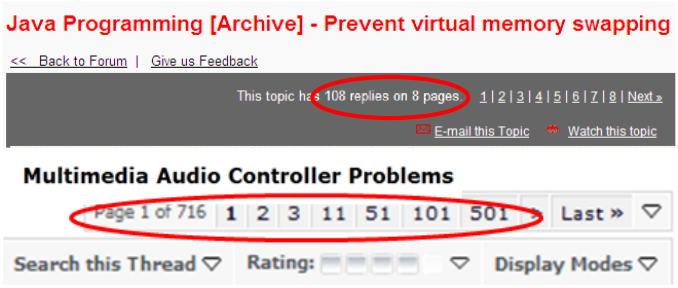

Fig. 2. Long Threads in Software Forums: Java (top; 108 posts in 8 pages), Multimedia (bottom; 10,737 posts in 716 pages)

proposing an approach to retrieve relevant answers to natural language queries, which are embedded in the mass of posts in software forums, via a tag inference approach.

In this study, as the first step of our framework we propose an engine to automatically infer tags for posts in software forums. Our system could be used to aid other works requiring the tagging of software forum posts. For example, Hou et al. extracted questions from news groups and tagged them manually [11]. Our system could potentially be used to provide the tags automatically. Treude and Storey shows the usefulness of tagging in software engineering activities to link technical and social aspects in managing work items [29]. Al-Kofahi et al. infer tags from software artifacts in IBM Jazz [1]. Duan et al. propose methods to extract the questions semantically close to a queried question [8] and, Jeon at al. retrieve similar questions [14]. Different from the above studies, we infer tags from software forums and utilize the inferred tags to effectively retrieve relevant answers to user queries expressed in natural language.

In the data mining and information retrieval communities, there have been several recent works on the extraction of question and answer sentences from online forums [4], [15], [6], [10]. We extend their study by building a semantic search engine that leverages a tag inference engine to return relevant answers from software forums.

III. Data Source

In this section, we describe software forums, how data are extracted from them, and some properties of software forum dataset.

A. Software Forums

Software forums usually consists of many threads. Some threads are organized in a hierarchical fashion. Each node in a hierarchy corresponds to a particular domain of interest. Questions posted are varied; some ask about device drivers needed, libraries needed to accomplished a particular programming task, help to solve a particular bug, and many more. An example of a software forum with long threads is shown in Figure 2.

We crawl the webpages corresponding to the threads in the forums to collect our dataset. We make use of WinHTTrack [12] to crawl. The crawled pages contains raw content and they need to be cleaned. We process them by pruning the HTML tags and keep the most important part of the posts. This helps to reduce the noise generated by such HTML tags which will not be useful for tagging posts.

B. Data Properties and Challenges

In our observations, the posts can be classified into various types like questions, clarifying questions, answers, clarifying answers, feedbacks and some junks. The feedback could be classified as positive or negative. "I still get the same error", is an example of a negative feedback. Table 1 shows an example of each of the post types that are used in our study.

We have observed many peculiar behaviors in the threads different from a typical one shown in Table I. For example, a question can follow another question and there is no necessity that the question-conversation is completed. An example of an interwoven question is shown in Table II. This makes the task of tagging the posts harder. To address this we use the author's name along with the textual content of the posts. We observe that a forum often has authors that frequently ask clarifying questions and give answers whereas questions are posted mostly by new authors. Another challenge here is to find the independent questions in the same thread.

We also observe that the same question could be asked multiple times. For these, some users might have the patience to answer or they might just reply back saying "This question has already been answered. Please read old posts.". The challenge is to tag such post as junk.

Furthermore, a question might not be answered by any subsequent posts or could be answered by multiple users. Some answers might not work and the conversation continues with more clarifying questions, answers, feedbacks, and so on. The general conversation behavior is represented in Figure 3. Two typical forum conversation chains are shown in Figure 4.

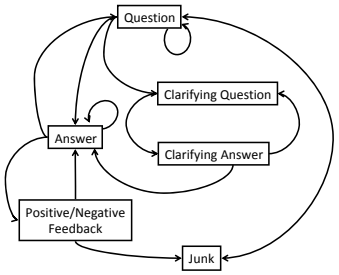

Fig. 3. Question-Conversation Graph

IV. Proposed Tag Inference Engine

In this section, we present our engine for tagging software forum posts which could be used to help in building an effective search engine.

Our engine is based on a text classification approach. As with any classification problem, we first need to clearly define our class tags. We defined 7 tags including: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk. We believe these tags well characterize the different types of post in software forums. The relationships among posts having the seven class tags are illustrated in Figure 3 and some example posts belonging to each class are shown in Table I.

TABLE I
A TYPICAL CONVERSATION

Post Type	Author	Post Content
Question	Lisa	I have a JTable with 4 columns. The first column is a jCheckBox which is working fine. The other three are JComboBoxes. They work fine when you click them and select from the list, but our end users only want to navigate with the keyboard. Below is the code for my table. Any help .. code snippet..
Clarifying questions	Christiaan	When you tab to the component, does it become active .....
Clarifying answer	Lisa	When I tab to the combobox it doesn't seem to have focus on the component unless I hit the F2 key.
Answer	Christiaan	Hi Lisa, I think your best bet is to add KeyListener to the table addKeyListener(KeyListener l) and listen to certain keys and next call JTable.editCellAt(int row, int column) to put the cell in edit mode...
Clarifying question	Lisa	but Since I have four columns, the first being a checkbox which is working fine and the next three being ComboBoxes, would I need to check which column that is selected ....
Answer	Christiaan	Hi, I think the following code should help you: some code lines....
Positive Feedback	Lisa	THANK YOU, THANK YOU, THANK YOU!!!!!!!!!!!!!!!!!! It works great. You are a lifesaver. Lisa
Junk	Linda	I'm a newbie from Kansas. Really like this forum so far and am looking forward to contributing!

TABLE II
EXAMPLE OF INTERWOVEN QUESTIONS

Post Type	Author	Post Content
Question	Blu	Oke, i have the same problem, doesn't recognize the multimedia audio controller. Before I installed a fresh windows xp on my pc, i runned sandra (something like belarc) and it said under multimedia audio controller: intel 82801eb ich5 ac '97
Questions	Pink	I have the same problem after formatting my hard drive. I'll post the information i think is needed. OperatingSystem: Windows XP Professional Service Pack 2 (build 2600) System Model MEDIONPC
Answer	BW	Get the driver here http://www.realtek.com.tw/downloads/...&Software=True and install
Answer	BW	Intel http://downloadfinder.intel.com/scri...5&submit=Go%21 it's No:3 on the list, install and then restart your comp.

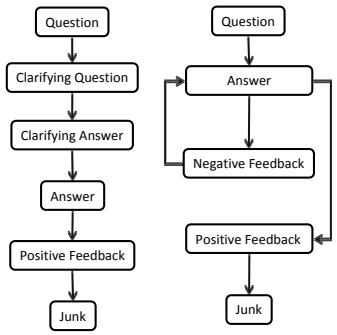

Fig. 4. Sample Forum Conversation Chains

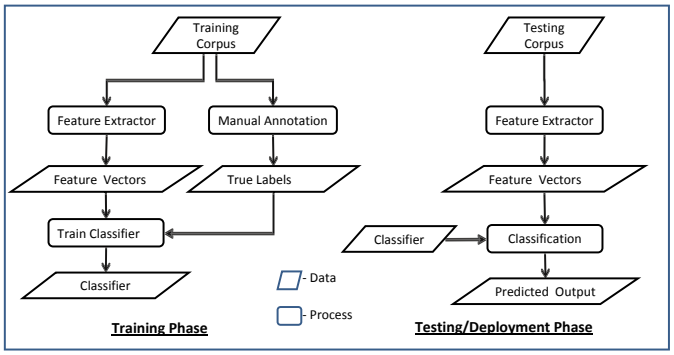

Fig. 5. Proposed Tag Inference Engine

All classification algorithms rely on a sufficient amount of labeled/tagged *training data*. For our task, it means for each of the post type that we have defined, we need to identify a set of posts that belongs to this type to be used as training data. To this end, we manually annotate the posts for each software forum we study according to our classification scheme.

Once tagged training data is created, the next step is to extract features from these examples. For most text classification problems, usually features simply include individual words from each piece of text to be classified. This is often referred to as a bag-of-words vector space model in information retrieval. Additionally, we also use author information as an additional feature because we observe that some authors can be highly correlated with certain types of posts. Auto labeling the authors according to expertise levels and use it as features may further benefit the model, which we keep for the future work.

After the training posts are transformed into feature vectors based on the features we choose to use, we can then apply a classification algorithm to train a classifier.

Figure 5 shows the structure of our tag inference engine. As can be seen, at the training stage, the posts are manually tagged and processed into feature vectors. The learning algorithm makes use of these tagged feature vectors to train a classifier. This classifier is later used in the classification stage on unseen new posts.

In the following sub-sections, we describe some components of our tag inference engine in more detail. We first describe our feature extractor component, followed by our classification model.

A. Feature Extractor

Most classification algorithms require data to be represented as feature vectors. We use the following steps to convert each post into a feature vector.

Stopword Removal: Stopwords are non-descriptive words such as prepositions (e.g. *of*, *in*) and pronouns (e.g. *he*, *it*), and are usually removed. We use a standard stopword list obtained from www.ranks.nl/resources/stopwords.html that contains a long list of 671 stopwords.

Stemming: Stemming is another commonly used technique in information retrieval. It normalizes words with the same root by removing certain suffixes of the words. For example, *computing*, *computer* and *compute* can all be normalized into the stem *comput*. We used Porter's stemming algorithm [26] to process our text.

Author Information: Each post is associated to one author. Some authors tend to provide answers while others questions, and yet others, provide junk like advertisements or irrelevant information. Hence considering author information for tagging of posts would be beneficial.

Post Content: For text classification, usually individual words are used as features. We use the word/term frequency (i.e., the number of times the word appear in the post) as the weight of each feature.

In this study, we are interested on 4 different feature extractor configurations:

1) STOP: Only the textual content of the posts, with stop words being removed, are considered as features.
2) STOPA: Both the textual content of the posts, with stop words being removed, and author information are used as features.
3) STEMA: Both the textual content of the posts, with stemming being performed, and author information are used as features.
4) SSA: Both the textual content of the posts, with stop words being removed and stemming being performed, and author information are used as features.

B. Classification Models

We consider two classification models: Independent post classification and context dependent post classification. We describe each of the classification model that supports our tag inference engine in the following paragraphs.

1) Independent Post Classification: In independent post classification, we treat each post independently as shown in Figure 6. T_i represents a thread in the software forum. P_0, P_1 and P_2 represent observed posts within T_i.L_0, L_1 and L_2 represent the hidden labels for these posts. Using independent post classification, each post is classified based on its content alone. Since our classification scheme has more than two classes, we perform multi-class classification for classifying the posts into 7 classes.

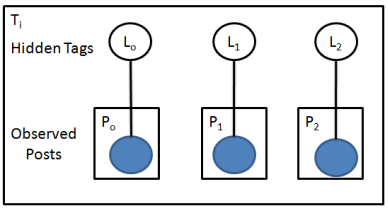

Fig. 6. Independent Post Classification

2) Context Dependent Post Classification: While a straightforward approach is to classify the posts separately, for our task we observed that there is a potential dependence between the class tags of consecutive posts. For example, a question is likely to be followed by either a clarifying question or an answer rather than a feedback message. This kind of dependence can be naturally modeled by a Hidden Markov Model as shown in Figure 7. In this approach we use the content together with the context to classify the posts. The context of a post is its k immediate preceding posts. Our intuition is that this approach could have better accuracy than independent post classification. We empirically investigate whether this is the case in our experiments described in Section VI.

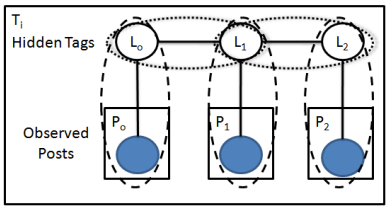

Fig. 7. Context Dependent Post Classification. Context dependent post classification model utilizes two pieces of information for tagging: correlation between the contents of the observed posts and specific tags (shown in the figure by dashed circles), and correlation between neighboring tags (shown in the figure by dotted circles)

V. Proposed Search Engine Framework

In this section, we describe how we could utilize inferred tags to help the retrieval of relevant answer posts from software forums. We first describe a typical search engine framework. Next, we describe how we could embed our tag inference engine to form a semantic search engine framework.

A standard search engine framework, as shown in Figure 8, follows the following steps for processing a natural language query and retrieving relevant documents:

1) **Pre-processing:** The pre-processor takes in a set of raw documents and extracts a set of relevant features from them. The features are in the form of words existing in the document.
2) **Indexing:** The indexing tool builds an index from a collection of documents. This index would be used during the retrieval process to quickly locate relevant documents.
3) **Query processing and retrieval:** The retrieval tool processes user queries and retrieve relevant documents using the index. Top-k matching documents are returned along with their respective relevance scores.

Our proposed search engine model is shown in the Figure 8. This model consists of additional two components that retrieves the relevant answers to the query.

1) **Tagger:** We embed our tag inference engine into the standard search engine framework. As described in the previous sections, our tag inference engine (or tagger) would tag each document with various tags: question, answer, etc.
2) **Filterer:** This additional component filters the top-k matching documents from the retrieval tool based on their inferred tags. With the Filterer, relevant solution posts are kept while irrelevant posts, e.g., junk, are filtered out.

VI. Experiments

In this section we want to answer the following research questions:

RQ1 How accurate are the inferred tags?
RQ2 What is the best feature combination to be used?
RQ3 What is the best classification model to be used?
RQ4 Is our proposed semantic search engine framework leveraging inferred tags effective in retrieving relevant answers for users ?

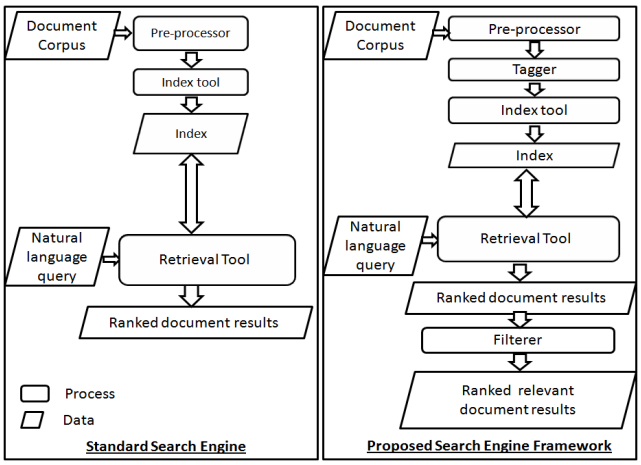

Fig. 8. Semantic Search Engine Framework.

$$Precision = \frac{tp}{tp + fp} \ , Recall = \frac{tp}{tp + fn}$$

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Fig. 9. Precision, Recall, F-measure. tp, fp, and fn corresponds to true positives, false positives, and false negatives respectively.

We answer the first three research questions in Section 6.3 and the last one in Section 6.4. We first describe the datasets that we use to evaluate our algorithm, and our evaluation criteria. We next describe our tag inference experimental study and the results of employing various configurations of our proposed approach in obtaining accurate tags. Finally, we describe our search engine experimental study that shows the effectiveness of our proposed semantic search framework in the retrieval of relevant answer posts for question queries expressed in natural language.

A. Experiment Settings

Our dataset is constructed by crawling webpages corresponding to posts from several software forums. We recursively followed the chain of replies to recover the posts from the threads in the forums. After the webpages have been crawled, we clean it and retrieve various information including the posts' message content and author. We analyze three different forums:

(Tips) www.softwaretipsandtricks.com
(DZone) http://forums.dzone.com
(Oracle) http://forums.sun.com/index.jspa

For Tips forum, we investigate the longer threads with at least 300 posts each. We focus on the multimedia group of threads. Tips forum consists of posts that involve more hardware and software related questions whereas the other two forums are programming related posts. DZone forum contains advertisement posts, e.g., "Pat Niemeyer, author of Learning Java by O'Reilly, lays out best practices with a modern FOSS enterprise Java stack.". Oracle forum consists of lengthy posts often with lengthy code snippets.

We perform two experiments: Tag inference and search engine enhancement. For the earlier, we evaluate the quality of our inferred tags as compared to manually labeled tags. For the latter, we evaluate the benefit of incorporating inferred tags to a standard document retrieval tool to improve the quality of the returned answers. We describe the settings for the two experiments below.

For the tag inference experiment, we manually tagged 6068 posts from the 3 software forums (4020, 680, and 1368 from SoftwareTipsandTricks, DZone, and Oracle, respectively). We use approximately half of the posts for training (i.e., 2000, 300, 620 posts from each of the 3 datasets are used respectively) and the rest for testing.

Search engine experiment is conducted on the same sets of posts to answer a set of 17 software queries. The complete list of queries is shown in the Table VII. The results from a standard information retrieval toolkit [1] and our semantic search engine framework, are consolidated for each query and are annotated by a team of five human annotators. The annotator would annotate 0, for irrelevant answer, 1, for partially answer, and 2, for the definite answer. The annotators for our experiments include three PhD students and two working professionals with 2-4 years of software development experience.

B. Evaluation Criteria

1) Precision, Recall, and F-measure: We evaluate our tag inference approach based on precision, recall, and F-measure. These measures have been commonly used to evaluate the accuracy of various retrieval, classification, and mining algorithms. Precision refers to the proportion of true positives over the sum of the true and false positives. Recall refers to the proportion of true positives over the sum of the true positives and false negatives. F-measure combines both precision and recall. This is useful as there is a tradeoff between precision and recall. The formulas for the evaluation criteria are shown in Table 9.

2) Mean Average Precision: We evaluate the quality of a search engine based on Mean Average Precision (MAP) [20]. A search engine returns an ordered list of documents for a given query. MAP gives a single numerical value to represent the quality of the ordering for relevance(1 or 2) or irrelevance(0).

The average precision value for a single query is calculated by taking the mean of the precision scores after each retrieved document is manually analyzed one by one for relevance. MAP is then the mean of the average precision scores over *a set* of queries. The formula for MAP is shown in the Figure 10.

3) Normalized discount cumulative gain (nDCG): Note that MAP can only handle cases with binary judgment: "relevant" or "irrelevant". Another evaluation measure called Normalized Discount Cumulative Gain (nDCG) has been proposed [16], which can handle multiple levels of relevance judgments. While evaluating a ranking list, nDCG follows two rules: First, highly relevant documents are more valuable than marginally

[1] http://www.lemurproject.org

TABLE III
OVERALL PRECISION, RECALL, AND F-MEASURE RESULTS (IN %)

	Classifiers							
Measure	HSSA	HSTEMA	HSTOPA	HSTOP	MSSA	MSTEMA	MSTOPA	MSTOP
Prec	67	66	62	62	50	50	50	49
Recall	71	70	67	65	57	57	56	53
F-measure	69	68	64	63	53	54	53	50

$$\mathrm{AvgP_i} = \int_{j=1}^{N} \frac{\mathrm{P}(j) \times \mathrm{pos}(j)}{\text{number of positive instances}}$$

$$\mathrm{P}(j) = \frac{\text{number of positive instances in top } j \text{ positions}}{j}$$

$$MAP_m = \frac{AvgP_i}{m}$$

Fig. 10. Average Precision and Mean Average Precision. $\mathrm{AvgP_i}$ is the average precision of query $\mathrm{q_i}$, *j* is the rank, *N* is the number of instances retrieved, pos(*j*) is either 0 or 1, where 1 represents relevant document and 0 represents irrelevant document. P(*j*) is the precision at the given cut-off rank *j*. MAP_m is the Mean Average Precision of the total number of queries m.

relevant document and Second, the lower ranking position a document (of any relevance level) has, the less valuable it is for the user, because it is less likely to be examined by the user. According to these rules, the nDCG value of a ranking list at position n is calculated as follow:

$$nDCG_p = IDCG_p \int_{i=1}^{p} \begin{cases} 2^{\mathrm{rel_i}} - 1 & , \quad \mathrm{i} = 1 \\ \frac{2^{\mathrm{rel_i}} - 1}{\log 2(\mathrm{i})} & , \quad \mathrm{i} > 1 \end{cases}$$

where rel_i is the rating of the i-th document in the ranking list, and the normalization constant ideal IDCG, $IDCG_p$ is chosen so that the perfect list gets a nDCG score of 1. In order to calculate nDCG, we use three ratings $0, 1, 2$, from annotation.

C. Tag Inference Experimental Results

We evaluate all four different feature set extractor configurations mentioned in Section IV. In the first configuration, we perform stopword removal (STOP). In the second configuration, we perform stopword removal and consider the authors of the posts (STOPA). In the third configuration, we perform stemming and consider the author of the posts (STEMA). In the fourth configuration we perform stopword removal, stemming and consider the author as well (SSA). We evaluate the two classification models: independent post classification and context dependent post classification. For the concrete implementation of the model, we use Multi-class Support Vector Machine (M) [21] for the first model and Hidden Markov Support Vector Machine (H) [9] for the second model. Thus, we have 8 different combinations: MSSA, MSTEMA, MSTOPA, MSTOP, HSSA, HSTEMA, HSTOPA, and HSTOP.

The overall result over all the datasets is shown in the Table III. The result broken down to each of the 3 datasets is shown in Table IV. The result broken down to each tag type (i.e., question, answer, clarifying question, etc) is shown in Table V. The following paragraphs describe the results in more detail.

From the overall result shown in Table III, it could be noted that the classification accuracy ranges from 50% to 69% (in terms of F-measure). MSTOP performs the worst while HSSA performs the best. In general considering author information improves accuracy. Also stemming seems to be more effective than stop word removal in improving the classification accuracy. The best combination is to consider author information, removing the stop words, and performing stemming (i.e., SSA). In general, for the same feature extractor configuration, Hidden Markov Support Vector Machine (SVM) performs better than multi-class SVM. Hidden Markov SVM takes into consideration the context of a particular post.

From the per dataset result shown in Table IV, it could be noted that the F-measure result ranges from 64% to 72% for HSSA method. The accuracy values are relatively stable across datasets. The result for Dzone has lower F-measure as compared to the other datasets. The poor performance is attributed to the fact that about 10% of the threads in this dataset contains merely junk (e.g., advertisements) rather than questions and answers.

From the per tag result shown in Table V, it could be noted that our approach works best in tagging questions and answers (which are the most important tags). Using HSSA, we could achieve up to 85% accuracy (in terms of F-measure) in tagging the questions. We could achieve up to 75-76% accuracy in tagging the answers (using HSSA and HSTOPA). It is interesting to note that for tagging feedback the performance of Hidden Markov SVM far outperforms that of multi-class SVM. This is the case as the detection of feedbacks often relies on information from the surrounding context of a particular post. In general, clarifying question could be detected better than clarifying answer. We could also identify the junks well, with an accuracy of up to 70% (in terms of F-measure, using Hidden Markov SVM).

Confusion Matrix. Table VI shows detailed information on the performance of HSSA in the form of a confusion matrix [20]. The rows correspond to the actual tag class (based on manual annotation) and the columns correspond to the predicted tag class. The values in the cells of the confusion matrix indicates the proportion of posts of a particular tag that are predicted/classified as of a particular tag. For example, the cell at row 6, column 1 with value 31 means that 31% of the negative feedback posts (row 6) was wrongly classified as question posts (column 1).

The percentages of correct classifications for a particular tag lie in the diagonal of the confusion matrix. The cell at row 1, column 1 with value 90 indicates that 90% of the questions were correctly classified. These values represent the true positives or correct predictions and all other off-diagonal values represent wrong predictions (either false positives or

TABLE IV
PRECISION, RECALL, AND F-MEASURE RESULTS FOR: SOFTWARETIPSANDTRICKS, DZONE, AND ORACLE (IN %)

		Classifiers							
Dataset		HSSA	HSTEMA	HSTOPA	HSTOP	MSSA	MSTEMA	MSTOPA	MSTOP
SoftwareTipsandTricks	Prec	73	72	76	73	63	63	61	58
	Recall	71	70	76	73	53	53	56	50
	F-measure	72	71	76	73	57	57	58	53
DZone	Prec	68	65	49	48	46	46	42	42
	Recall	61	60	42	44	35	37	33	32
	F-measure	64	62	46	46	40	41	37	36
Oracle	Prec	71	70	67	65	57	57	56	53
	Recall	67	66	62	62	50	50	50	49
	F-measure	69	68	64	63	53	54	53	50

TABLE V
PRECISION, RECALL, AND F-MEASURE RESULTS FOR: QUESTIONS (Q), ANSWERS (A), CLARIFYING QUESTIONS (CQ), CLARIFYING ANSWERS (CA), POSITIVE FEEDBACK (PF), NEGATIVE FEEDBACK (NF), JUNKS (J) (IN %)

		Classifiers							
Dataset		HSSA	HSTEMA	HSTOPA	HSTOP	MSSA	MSTEMA	MSTOPA	MSTOP
Q	Prec	89	87	88	88	90	90	88	87
	Recall	82	83	82	74	68	68	70	64
	F-measure	85	85	85	80	77	77	78	74
A	Prec	80	78	74	68	68	67	69	60
	Recall	71	70	80	60	63	63	58	40
	F-measure	75	74	76	64	65	65	63	48
CQ	Prec	64	64	65	65	44	57	18	4
	Recall	63	55	70	68	57	57	71	55
	F-measure	64	59	67	67	50	57	29	8
CA	Prec	28	40	48	48	9	13	15	4
	Recall	40	36	45	39	13	13	22	33
	F-measure	32	37	46	43	10	13	18	7
PF	Prec	54	51	54	42	0	0	1	3
	Recall	65	69	56	58	0	0	0	0
	F-measure	59	59	55	48	0	0	0	0
NF	Prec	6	57	33	21	0	0	2	3
	Recall	57	46	59	62	0	0	37	0
	F-measure	11	51	42	31	0	0	4	0
J	Prec	77	76	76	79	67	67	70	76
	Recall	65	63	60	60	53	54	60	51
	F-measure	70	69	67	68	59	60	65	61

TABLE VI
CONFUSION MATRIX

		Predicted Tag Class						
		Q	A	CQ	CA	PF	NF	J
Actual Tag Class	Q	**90**	5	1	3	1	0	1
	A	5	**80**	2	2	1	0	11
	CQ	3	20	**64**	4	0	0	9
	CA	23	17	17	**28**	3	1	10
	PF	10	19	0	3	**54**	1	13
	NF	31	48	1	4	3	**6**	6
	J	6	13	0	1	4	0	**77**

false negatives). The best predicted tags are question, answer, and junk. Clarifying question and positive feedback could also be classified reasonably well. The worst tags in terms of classification accuracy are negative feedback and clarifying answer.

Negative feedbacks are often predicted wrongly as questions (31%) or answers (48%) as forum users often add more information to their negative feedback posts to clarify what went wrong when they tried the proposed answer. Also, often some questions are added too in these negative feedback posts. For example, "Hi!Thanks, your idea is interesting (exec(..)), but it only works in shell, but not in a java aplication, even using exec Do you have any other ideas?xxdd" is a negative feedback containing a question. The accuracy of tagging clarifying answers is quite bad as they could be mis-tagged as questions (23%), answers (17%), clarifying questions (17%), or junk (10%). Again this is attributed to the fact that users tend to add questions in clarifying answers. For example, for a clarifying question, "Is FTP installed on the machine?", the reply is, "ftp is installed on the server and we are using JRE 1.3... on all client systems. Using ftp manually through a command prompt works fine. I'm wondering if I need a newer version of JRE", which is a clarifying answer together with a clarifying question.

Summary. As an answer to RQ1, we show that we could achieve an F-Score of up to 69% in inferring tags. As an answer to RQ2, our experiment shows that the best feature extractor configuration is SSA. SSA improves other configuration option by 1-6%. As an answer to RQ3, our experiment shows that context dependent post classification approach performs better than independent post classification by 11-17%.

D. Search Engine Experimental Results

To evaluate our overall search engine framework, we conduct a user-assisted experiments where users are tasked to label returned answers to 17 technical software-related queries shown in the Table VII that are expressed in natural language returned by a standard information retrieval toolkit and our proposed framework. We use the best configuration identified in our tag inference experiments, the HSSA configuration, for our tag inference engine.

We develop three search engines: Sys0, Sys1 and Sys2. Sys0 is created by treating all posts as individual documents in a collection. Sys1 is created with each thread treated as a single document. Sys2 is created in the same way as Sys1 but we also add the *inferred* tags for each post. We illustrate the setup for the three search engines in Figure 11. We build Sys0, Sys1, and Sys2 on top of Lemur [19], a language model toolkit that

TABLE VII
QUERIES TO EVALUATE SEARCH ENGINE PERFORMANCE

What is javax telephony InvalidArgumentException?
How to calculate Pi?
How to change this clock object to display 12 hour time instead of 24?
How to execute internet explore or windows word in java program?
What are the differences between a hash map and a hash table?
What is an invalid argument?
How to access CVS files?
How to set a combo box with different fonts?
How to generate random numbers or random booleans?
How to read files in Java?
How to generate prime numbers?
How to remove duplicates in an ArrayList?
How to get values from an ArrayList?
How to calculate the difference between two dates?
How to create a two dimensional array or 2D vector?
How to encrypt?
How to read a PDF file or PDF document?

TABLE VIII
NDCG AT POSITION P COMPARISON FOR 17 QUERIES.

SearchEngines	nDCG@1	nDCG@2	nDCG@3	nDCG@4	nDCG@5
Sys0	0.294	0.466	0.424	0.383	0.371
Sys1	0.206	0.422	0.341	0.323	0.315
Sys2	0.912	0.716	0.619	0.572	0.540

processes a query q and retrieves relevant documents from a set of documents. For Sys1 and Sys2, we convert the threads retrieved by Lemur back to posts. Sys1 retains all posts, while Sys2 only retains posts that are auto tagged as answers.

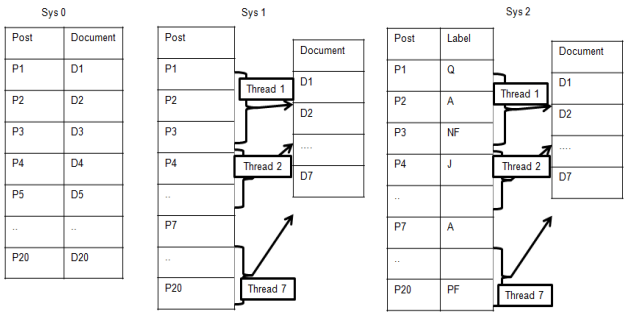

Fig. 11. System setup for search engine experiments. Sys0, Sys1 and Sys2 are the search engines under test. The labels for Sys2 are the tags automatically *inferred* by our inference engine.

We fed the 17 queries to all the engines and retrieved the top 20 posts for each query using Lemur. Finally we used MAP and nDCG to evaluate the results from each of the three engines against human annotations. The comparisons of the three systems in terms of MAP is shown in the Figure 12. It can be observed that Sys 0 performs better than Sys1 as many posts in a single thread might be junks or feedbacks rather than relevant answers. Sys2 with tagged posts, outperforms the other two. The tagging helps us to filter out the uninterested posts corresponding to feedbacks, junks, and so on from the retrieved results. With Sys2, we can display the relevant answers at the top of the returned list of posts which would benefit the users.

While, the results for nDCG as shown in the Table VIII, shows that Sys2 outperforms both Sys0 and Sys1 for all nDCG@1, nDCG@2, nDCG@3, nDCG@4 and nDCG@5.

Summary. As an answer to RQ4, we show that our framework could increase MAP from 17% to 71% and achieve nDCG@1 of 91.2%.

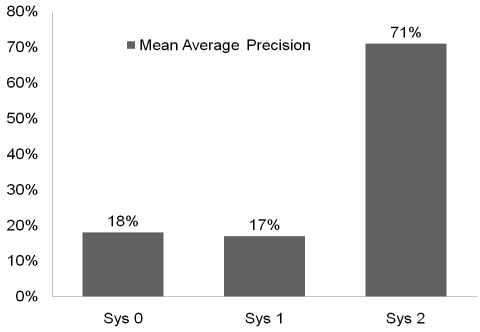

Fig. 12. MAP comparison for 17 queries.

E. Threats to Validity

Similar to other empirical studies, there are several threats to validity in interpreting the results.

Threats to construct validity corresponds to the appropriateness of our evaluation metrics. We use standard metrics: precision, recall, F-measure, and mean average precision. Thus, we believe there is little threat to construct validity.

Threats to internal validity correspond to the ability of our experiments to link the independent variable (i.e., input variable to be varied during the experiment) and the dependent variable (i.e., target variable). Since the real tags of the posts are not available, we need to tag the posts manually. Our manual tagging process might be prone to error. We have tried to minimize this error by performing some checks to the manually created tags.

Threats to external validity correspond to the ability to generalize our results. In this study, we have experimented with 3 sets of posts from different forums. A total of 6068 posts are investigated. We admit that this dataset is in no way near the number of all posts available in the various forums in the internet. In the future, we plan to extend our case study to include more posts, more forums and evaluate SE performance with questions on code. We also hope there would be more researchers interested in analyzing software forums and could help us in collecting and tagging more datasets.

VII. DISCUSSION AND FUTURE WORK

Some forums come with search utilities to help users in identifying relevant posts. It might seem that the search functionalities provided in these forums would suffice in helping users navigate through the mass of posts. Unfortunately, often there are many posts of various types (questions, answers, junks, etc) having similar keywords. Different keywords could also be used to convey the same meaning [31]. In this work, we study an orthogonal approach by inferring tags of the various posts and utilizing these tags to find relevant answers.

We provide not only question and answer tags, but also clarifying question, clarifying answer, positive feedback, negative feedback, and junk tags. These expressive and intuitive tags are useful. The tags could be used to extract answers which have been given positive feedback. Answers with only negative feedbacks can be pruned. Questions with no answers or answers receiving positive feedback can also be detected

and sent to experts. Clarifying questions and answers could be used to improve the search experience, e.g., by providing some help for users to refine his/her query. We leave these possible extensions for future work.

Posts are often highly ungrammatical and filled with spelling errors. In the future, we would look into the spelling variations to handle the noise in human written communication. The level of noise in software forum could at times be high. We also plan to explicitly incorporate technical terms, api (code questions) and jargons. We tested our model with 50% training data but, we would like to further explore the model accuracy with smaller training set to reduce the manual tagging task, as our future work.

Furthermore, it is interesting to combine our approach that works on macro (or post) level to latest research in information retrieval that works on micro (or sentence) level. It would also be interesting to develop an automated approach that could automatically arrange or cluster the forum posts in a hierarchical fashion to help users in finding the right answer to his/her question.

In this study, we only investigate 3 software forums. In the future, we want to extend this study further to investigate yet more forums to further validate the utility of our approach.

VIII. Conclusion

In this paper, we present a new approach to find relevant answers from software forums by leveraging an engine that automatically infers tags of posts in software forum threads. This tag inference engine automatically assigns 7 different tags to posts: question, answer, clarifying question, clarifying answer, positive feedback, negative feedback, and junk. We build a semantic search engine by leveraging the inferred tags to find relevant answers. Our experiments shows that our tag inference engine could achieve up to 67% precision, 71% recall, and 69% F-measure. The best result is obtained when using the SSA (Stop-Stem-Author) feature extractor and context dependent classification model implemented by Hidden Markov SVM. Our user-assisted study shows that as compared to a standard information retrieval approach, our proposed semantic search engine framework could increase mean average precision from 17% to 71% in retrieving relevant answers to various queries. We could achieve a Normalized Discounted Cumulative Gain (nDCG) @1 score of 91.2% and a nDCG@2 score of 71.6% on these queries.

Ackowledgements

We would like to thank Swetha Gottipati, Qui Minghui, Diao Qiming, Karthik Thirugnanam and Wang Shaowei for their help in the manual annotation process.

References

[1] J. Al-Kofahi, A. Tamrawi, T. Nguyen, H. Nguyen, and T. Nguyen. Fuzzy set approach for automatic tagging in evolving software. In *ICSE*, 2010.

[2] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *MSR*, 2006.

[3] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *CSCW*, 2010.

[4] G. Cong, L. Wang, C. Lin, Y. Song, and Y. Sun. Finding question-answer pairs from online forums. In *SIGIR*, 2008.

[5] F. C. de Sousa, N. C. Mendona, S. Uchitel, and J. Kramer. Detecting implied scenarios from execution traces. In *WCRE*, pages 50–59, 2007.

[6] S. Ding, G. Cong, C.-Y. Lin, and X. Zhu. Using conditional random fields to extract contexts and answers of questions from online forums. In *ACL*, 2008.

[7] B. Dit and A. Marcus. Improving the readability of defect reports. In *RSSE*, 2008.

[8] H. Duan, Y. Cao, C. yew Lin, and Y. Yu. Searching questions by identifying question topic and question focus. In *Proc. of Annual Meeting of the Association for Computational Linguistics- Human Language Tchnologies (ACL-HLT)*, 2008.

[9] www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html.

[10] L. Hong and B. D. Davison. A classification-based approach to question answering in discussion boards. In *SIGIR*, 2009.

[11] D. Hou, K. Wong, and H. Hoover. What Can Programmer Questions Tell Us About Frameworks? In *IWPC*, 2005.

[12] www.httrack.com.

[13] W. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. Hassan. Should I contribute to this discussion? In *MSR*, 2010.

[14] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *Procöf ACM Int. Conf. on Information and knowledge management (CIKM)*, 2005.

[15] M. Jeong, C.-Y. Lin, and G. Lee. Semi-supervised speech act recognition in emails and forums. In *EMNLP*, 2009.

[16] K. Jrvelin and J. Keklinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, Oct. 2002.

[17] H. Kagdi, M. Collard, and J. Maletic. An approach to mining call-usage patterns with syntactic context. In *ASE*, pages 457–460, 2007.

[18] H. Kagdi and D. Poshyvanyk. Who can help me with this change request? In *ICPC*, pages 273–277, 2009.

[19] www.lemurproject.org.

[20] C. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] svmlight.joachims.org/svm_multiclass.html.

[22] T. Nguyen, H. Nguyen, N. Pham, J. Al-Kofahi, and T. Nguyen. Graph-based mining of multiple object usage patterns. In *ESEC/SIGSOFT FSE*, pages 383–392, 2009.

[23] K. Pan, S. Kim, and E. W. Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14:286–315, 2009.

[24] P. Rigby and A. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *MSR*, 2007.

[25] M. Shevertalov and S. Mancoridis. A reverse engineering tool for extracting protocols of networked applications. In *WCRE*, pages 229–238, 2007.

[26] www.ils.unc.edu/∼keyeg/java/porter/PorterStemmer.java.

[27] M.-A. Storey. Beyond the lone reverse engineer: Insourcing, outsourcing, and crowdsourcing. In *WCRE*, 2009.

[28] S. Thummalapenta and T. Xie. Spotweb: Detecting framework hotspots and coldspots via mining open source code on the web. In *ASE*, pages 327–336, 2008.

[29] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development? In *ICSE*, 2009.

[30] http://twitter.com/.

[31] X. Wang, D. Lo, J. Jing, L. Zhang, and H. Mei. Extracting paraphrases of technical terms from noisy parallel software corpora. In *ACL/IJNLP*, 2009.

[32] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *ICSE*, pages 461–470, 2008.

[33] A. Wasylkowski and A. Zeller. Mining temporal specifications from object usage. In *ASE*, pages 295–306, 2009.

[34] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. In *ICSE*, 2009.

[35] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language api documentation. In *ASE*, pages 307–318, 2009.

[36] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Trans. on Software Engineering*, 31:429–445, 2005.