Ranking Crowd Knowledge to Assist Software Development

Lucas B. L. de Souza, Eduardo C. Campos, Marcelo de A. Maia
Department of Computer Science
Federal University of Uberlândia
Uberlândia, MG, 38400-902, Brazil
{lucas.facom.ufu, eduardocunha11}@gmail.com, marcmaia@facom.ufu.br

ABSTRACT

StackOverflow.com (SO) is a Question and Answer service oriented to support collaboration among developers in order to help them solving their issues related to software development. In SO, developers post questions related to a programming topic and other members of the site can provide answers to help them. The information available on this type of service is also known as "crowd knowledge" and currently is one important trend in supporting activities related to software development and maintenance.

We present an approach that makes use of "crowd knowledge" available in SO to recommend information that can assist developers in their activities. This strategy recommends a ranked list of pairs of questions/answers from SO based on a query (list of terms). The ranking criteria is based on two main aspects: the textual similarity of the pairs with respect to the query (the developer's problem) and the quality of the pairs. Moreover, we developed a classifier to consider only "how-to" posts. We conducted an experiment considering programming problems on three different topics (Swing, Boost and LINQ) widely used by the software development community to evaluate the proposed recommendation strategy. The results have shown that for 77.14% of the assessed activities, at least one recommended pair proved to be useful concerning the target programming problem. Moreover, for all activities, at least one recommended pair had a source code snippet considered reproducible or almost reproducible.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Technique—*Software libraries*
; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*

General Terms

Documentation, Experimentation

ICPC '14, June 2–3, 2014, Hyderabad, India

Keywords

Q&A services, crowd knowledge, recommendation systems

1. INTRODUCTION

In the last decade, concomitantly with the emerging of Web 2.0, a new software development behavior emerged and is changing the characteristics of software development. This change is the result of an extensive accessible structure of social media (wikis, blogs, questions and answers sites, forums) [18]. Similar to the way that open source development has changed the traditional process of software development [16], these new forms of collaboration and contribution have the potential to redefine how developers learn, preserve and share knowledge about software development.

One important example of social media is SO that is a Question and Answer (Q&A) website which uses social media to facilitate knowledge exchange between developers by mitigating the pitfalls involved in using code from the Internet. Its design nurtures a community of developers, and enables crowd sourced software engineering activities ranging from documentation to providing useful, high quality code snippets [1]. The set of information available on this social media services is called "crowd knowledge" and often become a substitute for the official software documentation [26].

Mamykina et al. conducted a statistical study of the entire SO corpus to find out what is behind the immediate success of it. Their findings showed that a majority of the questions will receive one or more answers (above 90% very quickly - with a median answer time of 11 minutes) [14]. In [26], Treude et al. pointed out that SO is particularly effective for code reviews, for conceptual questions and for novices.

SO has on its website, a search engine that allows users to query for textual content (for example: "how to sort a vector using Boost"). The result of a search is a set of discussions (threads), each one composed by a question and a series of answers. Users can sort threads according a number of criteria such as: the number of votes the question received; the relevance to the search query. However, considering only one criterion may bring some trouble to the user, for example: the search can return threads to the user that despite having relevance to the search query, have a negative rating by the community, or may return threads that despite being well voted by the community, are not very relevant to the developer's query. Thus, considering more than one criterion seems to be more appropriate in a recommendation strategy.

In this paper, we present a recommendation strategy that makes use of the information available on SO to suggest

question/answer pairs that may be useful to the programming task that a developer needs to solve. In this approach, we try to recommend pairs considering two aspects. The first concerns the textual similarity that the question/answer pair has with the task that the developer has at hand. The reason behind this criterion is that other developers may have had similar doubts in the past and posted questions on SO, so the answers to those past questions may be reused. The second criterion considers the score of the questions and answers from SO to try to recommend only Q&A pairs that were well evaluated by the crowd. The result of the recommendation process is a set of question/answer pairs that can be accessed by the developer via browser.

We made experiments to evaluate the recommendation strategy. The programming problems used in the experiments were extracted randomly from cookbooks for three topics widely used by the software development community: Swing, Boost and LINQ. The results have shown that for 27 of the 35 (77.14%) activities, at least one recommended pair proved to be useful to the target programming problem. Moreover, for all the 35 activities, at least one recommended pair had a reproducible or almost reproducible source code snippet.

The rest of this paper is organized as follows. Section 2 presents a Logistic Regression classifier used in this study to classify Q&A pairs into domain categories. Section 3 presents the experimental setting of our work. In Section 4 we present the results that are discussed in Section 5. In Section 6 we present the Related Work. In Section 7 we draw our conclusions.

2. CLASSIFICATION OF Q&A PAIRS

On SO, users ask many kind of different questions. Accordingly to Nasehi et al. [17] "SO question types can be described based on two different dimensions. The first dimension deals with the question topic: It shows the main technology or construct that the question revolves around and usually can be identified from the question tags that the questioner can add to the question to help others (e.g., potential responders) find out about what the question is about". Thus, if our goal is to recommend Q&A pairs for the topic Swing, we only consider in our approach Q&A pairs belonging to threads of discussion in which the question has the tag "swing" among its tags (a question on SO can have up to five tags). Yet accordingly to Nasehi et al. "questions from SO can also be classified in a second dimension that is about the main concerns of the questioners and what they wanted to solve". In this dimension, we considered the following five categories:

- *How-to-do-it*: The questioner provides a scenario and a question about how to implement it (sometimes with a given technology or API) [17];
- *Conceptual*: Conceptual questions on a particular topic (e.g., definition of concepts, best practices for a given technology). The questioner is waiting for an explanation of a particular subject or justification on certain behavior;
- *Seeking-something*: The questioner is looking for something more objective (e.g., book, tutorial, tool, framework, library) or more subjective (e.g., an advice, an opinion, a suggestion, a recommendation);
- *Debug-corrective*: Questions that deal with problems in the development code, such as errors at run time, notifications or unpredictable behavior. The questioner usually looks for revision in his code;
- *Miscellaneous*: The questioner has many different interests. Thus, he asks several questions. This usually leads to a mixture between the other categories (e.g., the questioner may be looking for a book and want a recipe for a problem).

The *How-to-do-it* category is very close to scenario in which a developer has a programming task at hand and need to solve it. For this reason, in our approach, we only consider Q&A pairs that are classified as *How-to-do-it*. In order to automate the selection of this kind of pairs, we developed a classification strategy to obtain only that type of Q&A pairs, which is shown on subsection 2.1.

2.1 Classification Algorithm

We performed a comparison between different classification algorithms to find the one that best classifies Q&A pairs from SO. In the classification process, we used seven classification algorithms: Logistic Regression (LR) [19, 11], Naive Bayes (NB) [13], Multilayer Perceptron (MLP) [9], Support Vector Machine (SVM) [13], J4.8 Decision Tree (J4.8) [13, 24], Random Forest (RF) [12] and K-Nearest Neighbors (KNN) [13].

We decided to classify the Q&A pair instead of classifying only the question body because we observed that in some cases the answer body provides relevant information to help to make decision of the Q&A pair category (e.g., differentiate between pairs of *How-to-do-it* and *Debug-Corrective* categories).

2.2 Definition of Attributes

We defined 10 attributes to characterize SO Q&A pairs. Out of these 10 attributes, 6 are related to the number of occurrences of terms, that are in a predefined keywords set, in the "title", "question body" and "answer body" of a pair. The remaining 4 attributes are boolean attributes and they are related to the presence or absence of source code or links in the question body and answer body for a given pair. We carried out a Feature Selection with Information Gain [28, 7] Filter to reduce the feature space and eliminate the least relevant attributes. The filter process selected 5 attributes.

We adopted a weighting criterion because there are keywords that seems to be more important than others. Thus, each keyword has its weight (1 or 5). Keywords with weight 1 are less important and are counted only once, while the keywords with weight 5 are more important and are counted 5 times when appeared in a Q&A pair.

The attributes that were selected using Information Gain Filter are shown below in decreasing order of Information Gain Value (the higher the value of Information Gain, the better the attribute contributes to the classification process):

- conceptualQty: Terms and expressions used to describe a pair in the category *Conceptual* (e.g., Weight 1: "lesson", "understand", "how much", etc.; Weight 5: "explain", "clarify", "difference between", etc.);
- lookingForQty: Terms and expressions used to describe a pair in the category *Seeking-something* (e.g., Weight

1: "idea", "suggestion", "guide", etc.; Weight 5: "looking for", "searching around", "seeking", etc.).

- howQty: Terms and expressions used to describe a pair in the category *How-to-do-it* (e.g., Weight 1: "algorithm", "implement", "function", etc.; Weight 5: "how to", "how can", "how does", etc.);

- answerHasCode: Boolean value that indicates whether exists source code in the answer.

- questionHasCode: Boolean value that indicates whether exists source code in the question.

2.3 SO Dataset

We downloaded a release of SO public data dump[1] (the version of March 2013) and imported the data into a relational database in order to classify the SO Q&A pairs. The "posts" table of this database stores all questions posted by questioners in the website until the date the dump was built. This table also stores all answers that were given to each question, if any.

We randomly selected from this relational database a batch of 400 Q&A pairs and manually classified them. In the classification process of each pair, we considered five categories:

Table 1 shows the results of manual classification performed on 400 selected Q&A pairs.

Table 1: Manual classification of 400 selected Q&A pairs.

Category	#Classified Q&A pairs
How-to-do-it	109
Conceptual	106
Seeking-something	121
Debug-Corrective	10
Miscellaneous	54

As the number of Q&A pairs of the *Debug-corrective* category was only 10, this category was not considered in the construction process of the dataset. Until the moment, no practical application was found for *Miscellaneous* category. Therefore this category was not considered in the construction process of the dataset. Obviously, all Q&A pairs classified as *Debug-corrective* or *Miscellaneous* were not used for training/testing the classifiers.

In the next step, we generated a ARFF file (Attribute-Relation File Format), containing the labeled instances and the information of attributes. This file was loaded into Weka [8] interface for the tests. We conducted an experimental study with 336 SO Q&A pairs divided into three domain categories: *How-to-do-it*, *Conceptual* and *Seeking-something*. The experiments were performed with Weka using a 10-fold-cross validation technique.

The best results were obtained with a Logistic Regression (LR) classifier with an overall success rate of 76.19% and 79.81% on *How-to-do-it* category. Table 2 shows the classification results obtained and has the following structure: Classifier, Success Rate (%), Correctly Classified Instances (Correct) and Incorrectly Classified Instances (Incorrect). In this work, we used the LR classifier to find out which are Q&A pairs of *How-to-do-it* category.

[1]http://blog.stackoverflow.com/category/cc-wiki-dump/

Table 2: Results with selected attributes.

Classifier	Success Rate	Correct	Incorrect
LR	76.1905%	256	80
NB	72.9167%	245	91
MLP	75.8929%	255	81
SVM	70.2381%	236	100
J4.8	69.6429%	234	102
RF	71.7262%	241	95
KNN (k = 5)	69.9405%	235	101

3. EXPERIMENTAL SETTING

In this section, we state our research goal, present the three topics considered in the experiments, explain how our recommendation approach works and details the experiment design and evaluation.

3.1 Research Goal

The aim of this paper is to recommend Q&A pairs to assist developers in their development tasks, considering that the recommended Q&A pairs should have textual similarity with the development task and they should be well evaluated by the SO community. A thread of discussion on SO is formed by a question and a series of answers to that question. We decided to recommend Q&A pairs instead of entire threads because the answers for the same question can have different scores, i.e., some answers can be better than others. We consider that a score of a pair (i.e., the number of upvotes minus the number of downvotes) is a indicative of its quality, because it's the main way that the SO community has to evaluate its content. So, we expect that recommended pairs are highly relevant in the context of the user task.

3.2 Considered Topics

We conducted our experiments on three topics for different programming languages (Java, C++ and .NET languages, respectively) widely used in the software development industry: Swing, Boost and Language Integrated Query (LINQ).

Swing is a toolkit created to enable enterprise development in Java. Therefore, developers can use Swing to create large-scale Java applications with a wide array of powerful components [6, 5].

Boost is a collection of C++ libraries. Each library has been reviewed by many professional developers before being accepted to Boost. Libraries are tested on multiple platforms using many compilers and the C++ standard library implementations [20].

LINQ (Language Integrated Query) is a Microsoft .NET framework programming model, which adds query capabilities to the .NET programming languages. These extensions provide shorter and expressive syntax to manipulate data [10].

3.3 Index Construction

We used the search engine Apache Lucene [2] to index the data. For a given topic (e.g., Swing) we obtain all threads from our database in which the question has a specific tag (e.g., "swing"). Then, from that set of threads, we obtain all Q&A pairs (e.g., if a thread has a question and n answers, we generate n Q&A pairs for that thread). Table 3 shows the number of Q&A pairs obtained for each topic considered in this paper.

The next step is to classify Q&A pairs in order to consider only *How-to-do-it* pairs. Table 4 shows the result of the pair's classification. For each Q&A pair classified as *How-to-do-it*, we remove its HTML tags, parse it, remove stop words and perform stemming on its content (text of title, question and answer, excluding the code snippets) using the Porter Stemming algorithm [22]. As questions and answers from SO can have source code snippets that are not appropriate to be parsed using the Lucene parser (because it is primarily a natural language parser) we treat those snippets in a different way. For Swing library we developed regular expressions to obtain the names of classes/interfaces/methods that are being created or called. For instance, suppose that the name of a method being declared is "addActionListener". As the Camel Case coding pattern is the most used pattern in Java, we split that names in its component terms. For the term "addActionListener" we obtain the words "add", "action" and "listener". We add into the Q&A pair document both the original term and its constituent terms, after performing stemming on them (in the example we would add to the Q&A pair document the terms "addActionListener", "action", "add" and "listener" (after removing stop words and stemming it). The reason to add to the document the name of classes/interfaces/methods being declared or used is that sometimes the asker wants to perform a specific task using a particular element of that API (e.g., "How to open a file using JFileChooser"), so including that information on the document could help our approach finding a pair that is adherent to those terms. For Boost we also developed regular expressions in order to identify the classes/methods/structs being declared or used. For LINQ, it is somewhat different because it is not a classical API: we checked if the source code snippet is using one of its operators (e.g., "OrderByDescending", "SelectMany") and perform similar processing as described for Swing on the name of those operators. The corpus of documents created is used to build a search index using Lucene. In the next subsection we present how we use this index to search Q&A pairs. For each API considered in this paper, we created an index following the previous approach. Table 5 shows the number of documents used to build the index for each topic. Observe that the number of documents is the same of Q&A classified in the category "How-to-do-it". This is explained because for each Q&A pair classified in that category is generated a document that composes the corpus used to build the index. Table 5 also shows the number of different terms in the documents for each topic.

Table 3: Total of Q&A pairs by topic.

Topic	Programming Language	Total of Q&A pairs
Boost	C++	14,558
Swing	Java	45,239
LINQ	C#	60,035

Table 4: Results of classification by topic.

Topic	*How-to-do-it*	*Conceptual*	*Seeking-something*
Boost	7,125	4,112	3,321
Swing	26,374	10,629	8,236
LINQ	39,592	13,962	6,481

Table 5: Indexes Information by topic.

Topic	Number of documents	Number of terms
Boost	7,125	55,383
Swing	26,374	187,914
LINQ	39,592	263,502

3.4 Searching in Lucene Indexes

The Lucene's index built for a topic can be used to search Q&A pairs that are relevant to a given query for that topic. We perform two types of search on this index, that we call *Scenario A* and *Scenario B*.

Scenario A corresponds to the situation where a developer has a task at hand, which will be solved using a particular API (e.g., "Boost"), but he does not know which API element (e.g., class or method) could help him solve his problem. For example, a developer could need to "read a text file using Boost". The title of the task (in this example "read a text file using Boost"), after being pre-processed (removal of stop words and stemming) is used as a search query to retrieve Q&A pairs.

Scenario B corresponds to the situation where a developer needs to solve a programming task using a particular element of the API. For example: someone could need to use the Swing library to "change the color of a JButton", where JButton is a widget class from Swing. In this case, the developer already knows which element has to be used.

We search Q&A pairs in *Scenario B* in a similar way of what we described for *Scenario A*. The difference is that we append to the task's title a string corresponding to a class name (in the case of Swing and Boost) or a operator name (in the case of LINQ) that we considered crucial in the solution presented in the cookbook for that problem. As we show later, the tasks considered in the experiments were extracted from cookbooks related to the topics. For example, for Swing one of the tasks selected for the experiment has the title "Action Handling: Making Buttons Work". The class "ActionListener" is important in the solution of the task. Thus, we append to the title the string "ActionListener", and the resulting string "Action Handling: Making Buttons Work ActionListener" is used as input to search on Lucene's index (after the removal of stop words and stemming of its content).

The result of a search on Lucene index is a ranked list of documents (i.e., Q&A pairs), in which the first one is the more similar to the search query and the last one is the less similar. Each pair in this ranking has a numeric value that we call *Lucene's score* that represents its similarity to the query. Thus, the first pair of the ranking has the greater *Lucene's score* and the last one has the smallest value.

3.5 Ranking Q&A Pairs by SO Score

Using the ranking returned for a search on Lucene's index, we can obtain pairs that have textual similarity with the input query but we cannot ensure nothing about its quality, i.e., among the pairs returned for a query may exist pairs well evaluated and pairs poorly evaluated by the SO community. Here we consider that a post's (question or answer) score is a proxy for its quality because the voting mechanism on SO is the main feature that allows SO members evaluate its content. Because each individual post on SO has its own score and our recommendation strategy will suggest Q&A pairs, being each pair composed by a question and an answer

to that question, we needed to define a metric that indicates the quality of the pair as a whole. One possible approach to achieve this, is to consider the mean value of the question's score and answer's score of a pair. However, we decided to consider the score of pair as the weighted mean value between the individual scores of its answer and question. We arbitrarily defined the values 7 and 3 for the weights of the answer and question from a pair. The reason to use this approach is because the answer seems to be more important than its belonging question, since the answer usually carries more information about the problem. We call this weighted mean as *SO score* of a pair.

Given a topic (e.g., Swing), we can calculate the *SO score* of each *How-to-do-it* pair that belong to that topic. In the next subsection, we will combine the *SO score* of a pair and its *Lucene's score* to build a ranking of pairs that will be used in our recommendation strategy.

3.6 Combining Scores to Rank Q&A Pairs

The *Lucene's score* of a pair indicates how much it is textually similar to a given search query, while its *SO score* indicates how much it was well voted by the SO crowd. Both of these aspects are considered in our recommendation, since we wanted to provide to the user of our system pairs that are at the same time related to the problem that he/she wants to solve and have good quality.

In order to combine both metrics in a single measure, we had to perform a normalization step. We normalize the *Lucene's score* of each pair returned for a query using min-max normalization technique. After this process, all pairs returned by a search on Lucene index have *Lucene's score* value in the range [0,1]. For those pairs, we also normalize it's *SO score* in the range [0,1]. The reason to normalize both of *Lucene's score* and *SO score* is because generally the *SO score* of a pair is much greater than its *Lucene's score*, i.e., those metrics have different nature. After this normalization step, we calculate the arithmetic mean of each pair. This mean is called *Final Score* and is used to rank the pairs in descending order. The top 10 pairs of this ranking are recommended to the user that queried the system.

3.7 Evaluation Criteria

In this section, we present the criteria used to evaluate the result of a recommendation made by our system. We defined two criteria in order to evaluate each pair recommended by our approach.

The first criterion is called **Relevance** (in short, *Relev*). This criterion is used to check to which extension the information contained in a pair can be used to help a developer solving the task that was queried on our system. The grade given in this criterion ranges from 0 to 4. The value 0 means that the recommended pair is not related at all to the task queried. The value 4 means that information contained in the pair can be used to completely solve the user's problem. This metric is not boolean because sometimes the information in a pair can be used to partially solve a problem.

The second criterion is called **Reproducibility** (in short, *Reprod*). This criterion is used to evaluate to which extension the source code snippets available on the question and answer bodies of a recommended pair can be easily compiled and executed. While the criterion *Relev* has a semantic aspect, i.e., its main goal is to check if the task can be solved using the information recommended, *Reprod* is a syntactic metric because it evaluates how easily the snippets can be compiled and executed, regardless if it is related to the search query at all. The grade also ranges from 0 to 4. The value 0 means that the recommended pair does not have source code snippets or its snippets cannot be compiled at all. The value 4 means that the snippets can be easily compiled and executed mostly without adaptation. This metric is not boolean because sometimes the pairs have source code snippets that although they cannot be directly executed, they could be compiled after some adjustments (e.g., many snippets are incomplete because they are missing a variable declaration, but if we declare the missing variable, the snippets becomes complete and could be compiled).

3.8 Experimental Design

We present an evaluation composed of experiments with the three considered topics to address the question whether our recommendation strategy can actually help developers in their development tasks. We consider a total of 35 development tasks: 12 for Swing, 12 for Boost and 11 for LINQ.

The Swing tasks were extracted from chapter 13 of Java Cookbook [5], that contains only tasks related to GUI (Graphical User Interfaces) technologies. There are 14 tasks on that chapter and we randomly selected 12 of them.

The Boost tasks were extracted from a Boost Cookbook [20]. We randomly selected 12 of 91 tasks available on this cookbook.

The LINQ tasks were extracted from a blog[2] developed by the Visual Basic Team from Microsoft. There are 12 tasks on that blog, however we only selected 11 of them because one task just had instructions on how to configure a database that is used in other tasks described on the blog, and thus it is not appropriate to be used in a experiment to recommend pairs for LINQ, since it is much more related to a generic database field than to LINQ.

We made a qualitative manual analysis of the recommendation for all 35 tasks. Figure 1 shows the design of our experiment. For each topic (Swing, Boost and LINQ) we made experiments to test *Scenario A* and *Scenario B*. From the 12 tasks previously selected for Swing, we randomly selected 6 for *Scenario A* and 6 for *Scenario B*. The same was done for Boost. For LINQ, as we had only 11 tasks, we randomly selected 6 and 5 tasks for *Scenario A* and *Scenario B* respectively. The input query for Scenario A was the title of the tasks (after stemming and removal of stop words). The input for *Scenario B* was a string formed by the title of a task appended with a name of a class (in the case of Swing or Boost) or operator (in the case of LINQ) that was important in the solution presented in the original cookbooks from where the tasks were extracted.

Table 7 shows for Swing, the 12 tasks selected for the experiment. The first 6 were included in *Scenario A* and the remaining 6 in *Scenario B*. The tasks for *Scenario B* are already shown with its title modified. For example, the title of the task 13.5 is originally "Action Handling: Making Buttons Work". In the table we present its title as "Action Handling: Making Buttons Work ActionListener", because "ActionListener" was the class name chosen to be append to the title, since it's a key class used in the solution for that problem. After removing stop words (if it has some) and stemming, the resulting string is used as a search query to

[2]http://blogs.msdn.com/b/vbteam/archive/tags/linq+cookbook/

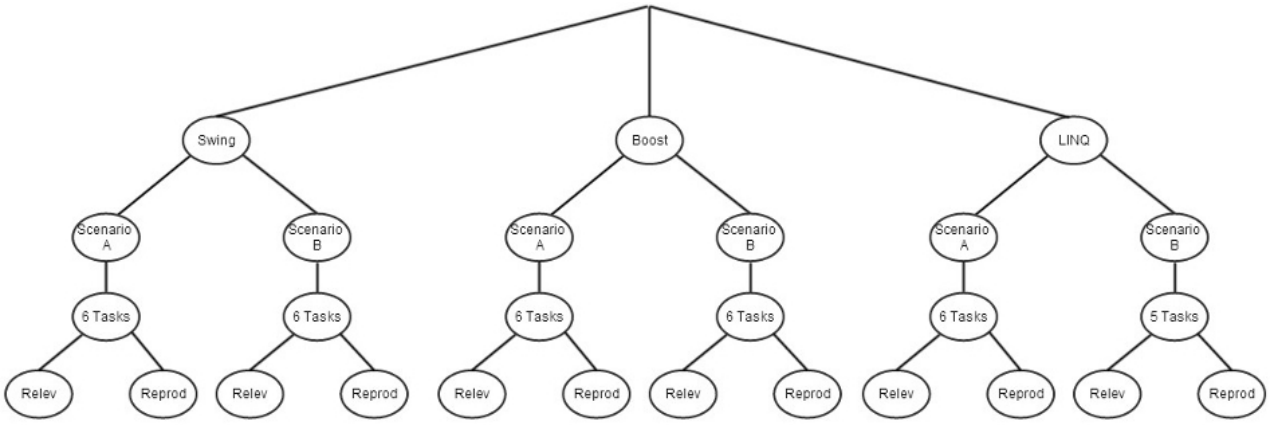

Figure 1: Experimental Design with distribution of the tasks per topic.

retrieve Q&A pairs. Similar tables are shown for Boost and LINQ (Tables 12 and 17 respectively). In those tables, the name of classes or operators used for *Scenario B* are shown in italic.

The first two authors of this paper (let's say *Author A* and *Author B*) individually evaluated for each of the 35 tasks, the 10 first recommended pairs. For each pair, they graded the two criteria previously described. In the Table 6, the column "Kappa Before" shows the Weighted Kappa [3] calculated to measure the agreement among the two evaluators. In that table, each row represents a triple "Topic/Scenario/ Criterion". Thus, in the first row the Weighted Kappa was calculated to compare the *Relev* grades given by the two authors for the pairs returned for the 6 tasks selected for Scenario A for Swing (so each row represents a comparison between 60 values, since the authors analyzed the first 10 pairs recommended for each task).

Table 6: Weighted Kappa - Agreement Comparison

Topic/Scenario/Criterion	Kappa Before	Kappa After
Swing/A/*Relev*	0.6	0.89
Swing/A/*Reprod*	0.84	0.95
Swing/B/*Relev*	0.58	0.92
Swing/B/*Reprod*	0.86	0.98
Boost/A/*Relev*	0.54	0.95
Boost/A/*Reprod*	0.94	0.95
Boost/B/*Relev*	0.81	0.94
Boost/B/*Reprod*	0.67	0.98
LINQ/A/*Relev*	0.95	0.97
LINQ/A/*Reprod*	0.81	0.93
LINQ/B/*Relev*	0.68	0.92
LINQ/B/*Reprod*	0.87	0.94

In a next step, the evaluations of the two authors were compared. The pairs in which the difference of the grades given by the authors was greater than or equal to 2 were marked for posterior discussion (e.g., *Author A* graded *Relev* as 4 for a pair and *Author B* graded *Relev* as 1 for the same pair). The reason to consider only the differences greater than or equal to 2 is because we don't consider a difference of 1 (e.g., *Author A* graded *Relev* of a pair as 3 and *Author B* graded it as 4) as a significant disagreement among the evaluators.

After marking those pairs with major divergence, the evaluators discussed each one and came to an agreement about them. After modifying the grades in this discussion step, the Weighted Kappa was calculated again and is shown in column "Kappa After" of Table 6. Comparing the values before and after this step, we can see that the agreement has been improved (a Weighted Kappa value "1" means a perfect agreement). Since we have obtained high overall agreement, we decided to consider only the evaluations made by *Author A*.

4. RESULTS

In this section, we present the results of the proposed experiment. Tables 8, 9, 10, 11, 13, 14, 15, 16, 18, 19, 20 and 21 have the same column structure: the task identifier and the first 10 ranking positions (P1 to P10). Each ranking position corresponds to a recommended pair. The tables show the ranking obtained for each task, considering the topic (Swing, Boost or LINQ), the scenario (A or B) and the criterion (*Relev* or *Reprod*).

For example in Table 8, in the first line of data, we have the results for task 13.14. We can observe that the best ranked pair (column P1) had received grade 2 (neutral). Moreover, we can see highly relevant pairs at P3 and P6.

We used the *Normalized Discounted Cumulative Gain* (NDCG) metric to have a numerical assessment of the ranking of pairs recommended in the experiments. NDCG is generally used to evaluate retrieval results from search engines and uses a multi-valued notion of relevance [15]:

$$NDCG(Q,k) = \frac{1}{|Q|}\sum_{j=1}^{|Q|} Z_{kj} \sum_{m=1}^{k} \frac{2^{M(j,m)}-1}{log_2(1+m)} \qquad (1)$$

k is the size of the result set. In our experiments $k = 10$, because we recommend 10 pairs to the user. *M(j,d)* is the metric value gave to document *d* for query *j*. Since we considered two criteria in our experiments, *M(j,d)* can be *Relev* or *Reprod*. We calculated NDCG value for both of those criteria. Z_{kj} is the normalization factor calculated such that NDCG is equal to 1.0. We followed the same approach used in a related work [21] to calculate this factor. In that approach, this factor is calculated in the scenario in which all documents retrieved have the maximum grade

Table 7: Swing Tasks

Task	Task Title
13.14	Program: Custom Font Chooser
13.13	Changing a Swing Program's Look and Feel
13.11	Choosing a Color from all the colors available on your computer
13.3	Designing a Window Layout
13.1	Choosing a File
13.8	Dialogs: When Later Just Won't Do
13.12	Centering a Main Window *JFrame*
13.2	Adding and Displaying GUI Components to a window *JFrame*
13.9	Getting Program Output into a *Window PipedInputStream*
13.4	A Tabbed View of Life *JTabbedPane*
13.5	Action Handling: Making Buttons Work *ActionListener*
13.6	Action Handling Using Anonymous Inner Classes *ActionListener*

Table 8: Swing - Scenario A (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
13.14	2	2	4	2	2	4	2	2	2	2
13.13	0	4	4	4	3	3	1	3	3	4
13.11	1	2	2	0	2	2	3	0	1	2
13.3	3	0	0	2	0	0	2	3	0	0
13.1	4	4	0	3	3	4	4	4	2	3
13.8	3	1	2	3	4	2	2	2	1	4

Table 9: Swing - Scenario A (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
13.14	0	0	4	3	0	3	4	3	2	0
13.13	3	0	0	4	4	0	0	0	4	3
13.11	0	0	0	3	3	3	4	4	3	0
13.3	0	4	0	0	0	0	4	0	0	0
13.1	4	4	4	4	4	4	4	0	3	3
13.8	0	0	0	3	4	3	0	4	4	4

Table 10: Swing - Scenario B (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
13.12	4	2	3	3	4	0	2	0	2	2
13.2	3	1	2	1	4	4	1	0	0	2
13.9	4	4	4	1	1	0	0	4	0	0
13.4	4	2	2	2	3	2	1	3	3	3
13.5	4	2	4	2	2	2	2	2	2	3
13.6	2	4	4	3	3	3	3	3	4	4

Table 11: Swing - Scenario B (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
13.12	0	0	0	4	3	3	4	0	2	0
13.2	4	0	0	0	0	0	4	0	2	2
13.9	4	4	4	3	4	4	4	4	0	0
13.4	4	0	0	3	4	4	0	4	4	4
13.5	0	4	0	0	0	4	0	0	0	3
13.6	3	4	3	3	4	4	4	4	4	4

Table 12: Boost Tasks

Task	Task Title
2.8	Parsing date-time input
3.1	Doing something at scope exit
12.5	Using portable math functions
12.7	Combining multiple test cases in one test module
10.7	The portable way to export and import functions and classes
3.5	Reference counting pointers to arrays used across methods
7.7	Using a reference to string type *string_ref*
10.2	Detecting RTTI support *type_index*
1.11	Making a noncopyable class *noncopyable*
9.2	Using an unordered set and map *unordered_set*
7.2	Matching strings using regular expressions *regex*
8.8	Splitting a single tuple into two tuples *vector*

Table 13: Boost - Scenario A (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
2.8	3	2	3	3	3	2	2	2	3	3
3.1	1	2	0	0	0	0	0	0	0	1
12.5	1	0	0	0	4	0	4	4	4	4
12.7	2	4	0	0	0	2	2	2	4	0
10.7	3	2	0	1	2	0	0	0	0	0
3.5	3	2	2	2	3	2	2	3	2	3

Table 14: Boost - Scenario A (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
2.8	2	4	4	4	4	0	4	4	4	4
3.1	4	4	4	0	4	1	0	2	0	0
12.5	0	0	0	0	4	0	4	4	0	0
12.7	0	0	0	3	4	0	0	4	3	4
10.7	4	0	4	0	4	0	3	4	4	0
3.5	4	0	4	0	0	4	0	0	0	4

Table 15: Boost - Scenario B (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
7.7	2	0	0	0	0	0	1	1	0	0
10.2	2	2	1	0	0	0	0	0	0	1
1.11	3	1	3	1	1	3	3	2	2	1
9.2	2	1	1	1	3	1	3	1	1	0
7.2	4	4	4	4	3	4	0	2	2	4
8.8	1	1	1	1	1	1	1	1	1	1

Table 16: Boost - Scenario B (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
7.7	4	0	4	1	4	2	4	4	4	4
10.2	4	0	0	2	4	4	0	0	0	0
1.11	0	3	0	4	2	0	4	4	4	2
9.2	0	4	0	3	4	4	3	4	2	2
7.2	0	4	4	4	4	4	4	4	0	4
8.8	4	4	4	4	2	4	3	3	3	4

Table 17: LINQ Tasks

Task	Task Title
2	Find all capitalized words in a phrase and sort by length (then alphabetically)
10	Pre-compiling Queries for Performance
1	Change the font for all labels on a windows form
5	Concatenating the selected strings from a CheckedListBox
11	Desktop Search Statistics
12	Calculate the Standard Deviation
3	Find all the prime numbers in a given range *Count*
7	Selecting Pages of Data from Northwind *Skip*
4	Find all complex types in a given assembly *Distinct*
9	Dynamic Sort Order *OrderByDescending*
8	Querying XML Using LINQ *Contains*

Table 18: LINQ - Scenario A (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
2	0	0	0	2	2	0	2	0	0	0
10	1	2	4	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
5	2	0	2	1	2	2	2	2	0	2
11	0	0	0	0	0	0	2	0	0	0
12	4	4	4	4	1	4	3	1	3	0

Table 19: LINQ - Scenario A (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
2	3	0	3	4	4	4	4	0	3	3
10	0	0	3	3	2	0	3	0	0	0
1	4	1	0	0	2	2	2	4	4	0
5	3	2	4	0	3	4	3	3	3	3
11	3	4	4	4	4	4	4	4	3	0
12	3	3	3	3	0	3	4	2	4	0

Table 20: LINQ - Scenario B (0 = Not Relevant, 4 = Highly Relevant)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
3	1	1	2	1	4	1	0	2	1	1
7	4	2	2	4	4	4	4	4	4	4
4	0	1	2	0	3	0	1	0	0	0
9	2	4	4	2	2	4	4	1	4	2
8	0	2	1	2	1	0	0	3	4	3

Table 21: LINQ - Scenario B (0 = Not Reproducible, 4 = Highly Reproducible)

Task	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
3	4	4	4	4	3	3	3	4	4	2
7	3	4	4	3	2	3	3	2	4	3
4	3	0	3	3	4	3	0	0	0	0
9	3	4	4	3	3	4	3	2	3	2
8	3	0	0	0	2	0	0	0	4	0

(value 4 in our case). We also decided to have an evaluation with this metric, because we could partially compare the results of both works (not totally because with use top-10 and they used top-15). However, using NDCG for this evaluation seems to be inadequate because only queries that had most of the 10 pairs with relevance close to four will have very high NDCG value, and this seems to be extremely stringent. We are mostly interested that there are some relevant pairs best ranked, i.e., not necessarily all 10 pairs need to be highly relevant because if the first encountered highly relevant pair resolves your problem, then it suffices. The normalization factor we calculated using this approach is $Z_{kj} \approx 0.01$. $|Q| = 35$ because we considered 35 tasks in the experiments.

The NDCG calculated was 0.3583 and 0.5243 for *Relev* and *Reprod* respectively.

Considering that it is desirable to have highly graded pairs in the top-10 pairs, we decided to analyze the number of pairs recommended by our approach having *Relev* or *Reprod* greater than or equal to 3. For this analysis, we use the graphics shown in Figure 2. The three graphics in the first line of the figure consider the criterion *Relev*, while the remaining three in the second line consider criterion *Reprod*.

Each graphic consider the activities selected for *Scenario A* (diamond symbol) and the ones for *Scenario B* (square symbol). The activities appear in the graphics in the order shown in Tables 7, 17 and 12. Thus, in the first graphic of the figure (graphic "Boost - *Relev*"), the first activity corresponds to "Parsing date-time input" (*Scenario A*) and "Using a reference to string type string ref" (*Scenario B*). Besides considering the value 4 for criterion *Relev* we also considered the value 3 because, although not being the best case, the pairs evaluated with grade 3 for *Relev* could be used to almost solve the entire problem represented by the search query. Similarly, pairs with *Reprod* equal to 3, had code snippets almost complete. As we can see in the graphics, all activities for Swing had pairs with *Relev* ≥ 3. Only eight of the 35 tasks didn't have pairs with *Relev* ≥ 3 among the 10 recommend pairs. All the 35 activities had at least one pair with *Reprod* ≥ 3.

5. DISCUSSION

We partially compared our results with the results presented by Ponzanelli et al. [21] to evaluate our work. In their work they developed a plugin for Eclipse IDE called SEAHAWK in order to recommend content from SO to help developers solve programming problems. The main differences of both works are:

- Our recommendation approach considers two aspects to suggest content: the textual similarity that the pairs have with the search query and the their score. In their approach, only the textual relevance is considered.
- The tasks used in both works are different: while in our approach we randomly selected the tasks considered in the experiment from cookbooks, in their paper they selected the activities from a Java programming course. When using tasks from cookbooks instead of a programming course, we focus more on practical tasks that a developer can face daily than on didactic items used to teach a topic.
- We performed a classification step to obtain the *How-to-do-it* pairs. Only those pairs are considered in the

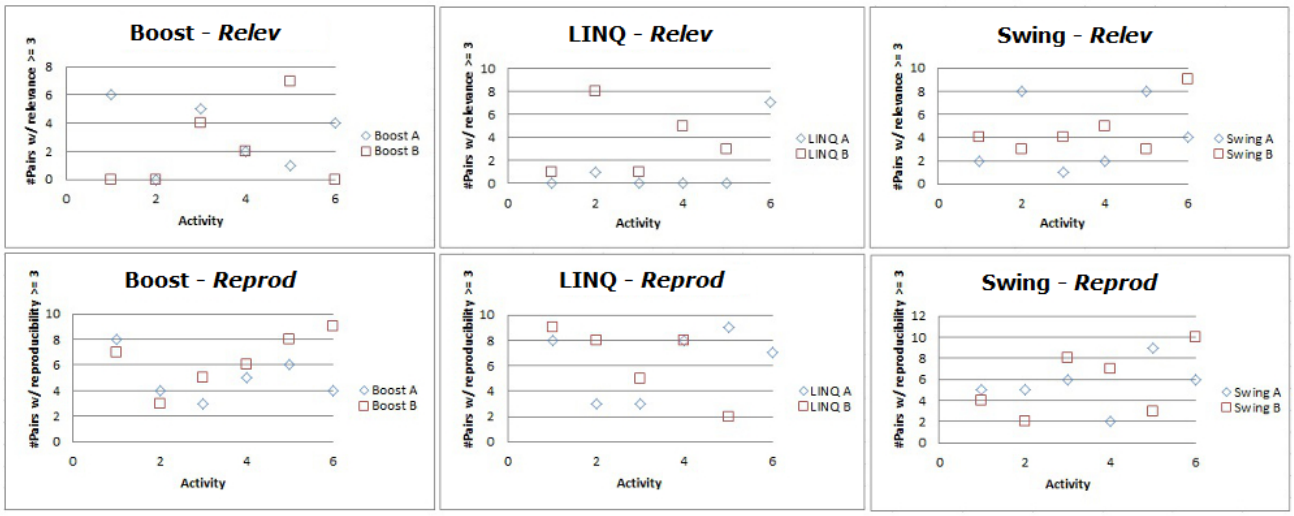

Figure 2: Numbers of pairs having Relev $\geq$ 3 (First Line) and Reprod $\geq$ 3 (Second Line).

recommendation process. Ponzanelli et al. consider all kind of questions in their recommendation approach. We consider this classification step important, since we can discard pairs that are more theoretical than practical (e.g., pairs in the categories *Conceptual* or *Seeking-something*).

- In their experiments, only Java tasks were considered. Here, we tested our approach using tasks from three different topics (Swing, Boost and LINQ) that are related to different programming languages (Java, C++ and .NET languages respectively).

- In their paper, they recommended entire SO threads. Here, we recommend individual Q&A pairs, since for a question can exist well voted answers and poor voted answers. The reasoning behind this approach is to recommend only the portions of a Q&A threads that are well evaluated by the SO crowd.

- We defined two different criteria that were used in our experiments Relevance and Reproducibility. Ponzanelli et al. only used the Relevance criterion.

- In our experimental design, we presented an evaluation made by two different subjects. Ponzanelli et al. did not present in their paper the manner in which their results were assessed (e.g., by one or two authors).

- Ponzanelli et al. developed a plugin for Eclipse. We intend in a future work implement our approach in the form of a plugin for an IDE.

- SEAHAWK recommends 15 Q&A threads. In the experiments presented, we recommend only 10 Q&A pairs, although this number could be adjusted to the user needs.

Although those many differences, the NDCG obtained for criterion *Relev* here is far superior than the one obtained by Ponzanelli et al. (0.3583 and 0.0907 respectively), which suggests that our approach outperforms theirs.

Analyzing Figure 2, we can see that for *Relev*, Swing obtained better results over Boost and LINQ since for all tasks it had at least one pair with *Relev* ≥ 3 for both *Scenarios A* and *B*. For LINQ, 2 of the 6 tasks had recommended pairs with *Relev* ≥ 3 in the case of *Scenario A* and for *Scenario B* all 5 tasks had at least one pair having *Relev* ≥ 3. For Boost, 5 tasks from *Scenario A* and 3 from *Scenario B* had pairs with at least one recommended pair having *Relev* ≥ 3.

There could be two main reasons to explain the low number of pairs with *Relev* ≥ 3 for some tasks. First, our approach uses the title of a task as a input in the search engine. Some tasks do not have a precise information on its title. For instance, the task "Desktop Search Statistics" for LINQ has the goal to search the file system of a computer and count the number of items that are documents, images or e-mails. Using only the title information, we cannot know that. In other words, some tasks do not have sufficiently specific title. Another reason that could justify that some tasks did not had recommended pairs with *Relev* ≥ 3 is the absence of information related to that task in the SO's crowd knowledge. For the 35 tasks, 27 had at least one recommended pair with *Relev* ≥ 3 and 19 had at least one recommended pair with *Relev* $= 4$. Maybe if we run the experiment in the future, the dataset will have more posts and those tasks could had been covered by the SO crowd.

All 35 tasks had at least one recommended pair with *Reprod* ≥ 3 and 34 tasks had at least one recommended pair with *Reprod* $= 4$, indicating that our strategy has good performance in recommending snippets that are reproducible or can become reproducible with minor adjustments. The two main reasons found that explains that difficulty to reproduce some source code snippets are:

- The use of a variable that was not declared. For instance, consider a recommended Q&A pair related to Swing API, whose answer has a code snippet that uses a widget object (e.g., a button), but does not show how to create the object. Although create a button in Swing is very simple for most people who have some experience in programming GUIs, it could be not trivial for someone new to GUI programming.

- The omission of some lines of code (e.g., some answers use "..." to indicate that some lines are omitted in a code snippet). Again, this lack of information makes it difficult to use the code snippet in a programming environment like an IDE.

Although not shown in Figure 2, 26 of the 35 tasks have at least one recommended Q&A pair which has both criteria *Relev*, *Reprod* ≥ 3. This is an important metric because there are no value about Q&A pairs which are very reproducible, but are not relevant to the developer problem.

5.1 Threats to Validity

The classification process of SO Q&A pairs into categories is subjective. Therefore, exist the possibility of disagreement among people about the category of a Q&A pair.

Another aspect to note is that Q&A pairs of *Miscellaneous* or *Debug-corrective* categories will be incorrectly classified by the classifier on one of the other 3 categories: *How-to-do-it*, *Conceptual* or *Seeking-something*.

The choice of keywords for each attribute of the classifier is also subjective, because each person could define a different keywords set. The values of the weights used to differentiate between the least relevant terms (weight equals to 1) and the most relevant terms (weight equals to 5) are arbitrary and different values could produce different classification results.

The values of the weights (7 and 3) used to calculate *SO score* of a pair were arbitrarily defined. However, we achieved reasonable results with this choice.

A qualitative analysis involving manual inspection of the recommendations was necessary to obtain rich interpretation. However, this detailed interpretation is subjective. Notwithstanding the inherent subjectivity of the process, many factors contribute to the robustness of the evaluation. The evaluation of the recommended pairs was done in two phases. First, each of the two evaluators made an individual assessment. Then, a consensual assessment was done to diminish the bias (i.e., the Weighted Kappa values improved after this discussion step).

Another threat to validity concerns the classes selected to compose the queries for task in *Scenario B*, since this is subjective and different choices could lead to different results.

6. RELATED WORK

Treude et al. [26] analyzed data from SO to categorize the kinds of questions that are asked and to explore which questions are answered well and which ones remain unanswered. Their preliminary findings indicate that Q&A sites are particularly effective for code reviews and conceptual questions. They analyzed the titles and body texts of 385 SO questions and found the following categories, ordered by their frequency: *how-to*, *discrepancy*, *environment*, *error*, *decision help*, *conceptual*, *review*, *non-functional*, *novice*, *noise*. They also posed questions regarding the impact of social media on software development knowledge, and how it could influence the habits of developers.

Ponzanelli et al. [21] presented an integrated and largely automated approach to assist developers who want to leverage the crowd knowledge of Q&A services. They implemented SEAHAWK, a recommendation system in the form of a plugin for the Eclipse IDE to harness the crowd knowledge of SO from within the IDE. This plugin automatically formulates queries from the current context in the IDE, and presents a ranked and interactive list of results. SEAHAWK lets users identify individual discussion pieces and import code samples through simple drag & drop.

Sawadsky et al. presented FISHTAIL [23], an Eclipse plugin which assists developers in discovering code examples on the web relevant to their current task. FISHTAIL suggests codes examples according to the most changed program entity's name. In some of the activities (*Scenario B*) used to test our approach we also focused on entities name (e.g., classes name) since this name was part of the search used to query our system. In our approach we perform code retrieval on SO. Since we rely on SO, the code samples are already assessed by the community. Thus, the developer has not to assess their validity.

HIPKAT [27] is a recommender system developed to support newcomers in a project by recommending items from problem reports, newsgroup, and articles. Our approach focus on recommend content from SO instead of providing resources from in-project knowledge.

Cordeiro et al. [4] presented an Eclipse plugin to help developers in problem solving tasks. Based on an exception's stack trace gathered from the IDE's console, they suggest related documents from SO. Instead of focusing on stack traces, we focus on the task title to query an index previously created and retrieve Q&A pairs.

Takuya et al. presented SELENE [25], a source code recommendation tool based on an associative search engine. It spontaneously searches and displays example programs while the developer is editing a program text. Our work also lies in the field of search engines, but we suggest Q&A pairs taken from SO to enrich the information provided by code snippets.

7. CONCLUSIONS

We presented a novel approach to leverage the Q&A crowd knowledge. This strategy recommends a ranked list of pairs of questions/answers from SO. The ranking criteria take into account the textual similarity of the pairs with respect the developer's problem and the quality of the pairs. We developed experiments considering 35 programming problems distributed on three different topics (Swing, Boost and LINQ) widely used by the software development community.

We made a qualitative manual analysis of the recommended Q&A pairs considering two criteria: **Relevance** and **Reproducibility**. We obtained a NDCG value of 0.3583 for the first criterion and 0.5243 for the second criterion. The results have shown that for 27 of the 35 (77.14%) activities, at least one recommended pair proved to be useful to the target programming problem. Moreover, for all the 35 activities, at least one recommended pair had a reproducible or almost reproducible source code snippet. These results suggests that our approach outperforms the results obtained in a related work [21].

As future work, the implementation of our approach in the form of a plugin for an IDE can disseminate the use of the approach.

8. ACKNOWLEDGMENTS

This work was partially supported by FAPEMIG grant CEXAPQ-2086-11 and CNPQ grant 475519/2012-4.

9. REFERENCES

[1] O. Barzilay, C. Treude, and A. Zagalsky. *Facilitating Crowd Sourced Software Engineering via Stack Overflow*, pages 297–316. Springer, New York, 2013.

[2] A. Bialecki, R. Muir, and G. Ingersoll. Apache lucene 4. In *SIGIR 2012 Workshop on Open Source Information Retrieval*, pages 1–8, 2012.

[3] J. Cohen. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit., 1968.

[4] J. Cordeiro, B. Antunes, and P. Gomes. Context-based recommendation to support problem solving in sof. development. In *Proceedings of 3rd Int. Workshop on RSSE)*, pages 85–89, 2012.

[5] I. F. Darwin. *Java Cookbook*. O'Reilly Media, Sebastopol, CA, USA, 2004.

[6] R. Eckstein, M. Loy, and D. Wood. *Java Swing*. O'Reilly Media, Sebastopol, CA, USA, 1998.

[7] G. Forman, I. Guyon, and A. Elisseeff. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software : An update. *SIGKDD Explorations*, pages 10–18, 2009.

[9] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, PTR Upper Saddle River, NJ, USA, 1998.

[10] J. Hilyard and S. Teilhet. *C# 3.0 Cookbook*. O'Reilly Media, Sebastopol, CA, USA, 2007.

[11] S. le Cessie and J. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

[12] V. Lempitsky, M. Verhoek, A. Noble, and A. Blake. Random forest classification for automatic delineation of myocardium in real-time 3D echocardiography. In *Functional Imaging and Modeling of the Heart*, pages 447–456. Springer Berlin Heidelberg, 2009.

[13] M. Linares-Vasquez, C. McMillan, D. Poshyvanyk, and M. Grechanik. On using machine learning to automatically classify software applications into domain categories. *Empirical Software Engineering*, pages 7–8, 2009. Published by Springer US.

[14] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2857–2866, New York, NY, USA, 2011. ACM.

[15] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[16] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 263–272. ACM, 2000.

[17] S. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example? A study of programming Q&A in Stack Overflow. In *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34, 2012.

[18] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. *Georgia Tech, Tech. Rep.*, 2012.

[19] M. Pohar, M. Blas, and S. Turk. Comparison of logistic regression and linear discriminant analysis : A simulation study. *Metodološki Zvezki*, 1(1):143–144, 2004.

[20] A. Polukhin. *Boost C++ application development cookbook*. Packt Publ., Birmingham, 2013.

[21] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In A. Cleve, F. Ricca, and M. Cerioli, editors, *CSMR*, pages 57–66. IEEE Computer Society, 2013.

[22] M. F. Porter. Readings in Information Retrieval. chapter An algorithm for suffix stripping, pages 313–316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[23] N. Sawadsky and G. C. Murphy. Fishtail: From task context to source code examples. In *Proceedings of the 1st Workshop on Developing Tools As Plug-ins*, pages 48–51, New York, NY, USA, 2011. ACM.

[24] L. Sehgal, N. Mohan, and P. S. Sandhu. Quality prediction of function based software using decision tree approach. In *International Conference on Computer Engineering and Multimedia Technologies (ICCEMT)*, pages 43–47, 2012.

[25] W. Takuya and H. Masuhara. A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development*, pages 17–20. ACM, 2011.

[26] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 804–807. ACM, 2011.

[27] D. ČubraniĆ, G. C. Murphy, J. Singer, and K. S. Booth. Learning from project history: A case study for software development. In *Proceedings of the 2004 ACM Conference on CSCW*, pages 82–91. ACM, 2004.

[28] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning (ICML)*, 1997.