JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
*J. Softw. Evol. and Proc.* 2016; **00**:1–32
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

# Searching Crowd Knowledge to Recommend Solutions for API Usage Tasks

Eduardo C. Campos, Lucas B. L. de Souza, Marcelo de A. Maia

*Faculty of Computing, Federal University of Uberlândia, Uberlândia, MG, 38400-902, Brazil*

SUMMARY

Stack Overflow (SO) is a question and answer service directed to 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.
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 question-answer pairs from SO based on a query. The criteria for ranking are based on three main aspects: the textual similarity of the pairs with respect to the query related to the developer's problem, the quality of the pairs, and a filtering mechanism that considers 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 *Lucene+Score+How-to* approach, 77.14% of the assessed activities have at least one recommended pair proved to be useful concerning the target programming problem. Copyright © 2016 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: 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 it 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) [1]. Similarly to the way that open source development has changed the traditional process of software development [2], 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, which is a notable example of technical question and answer (Q&A) service that gained popularity among developers and is an important venue for sharing knowledge on software development [3]. This service offers a web platform to programmers for discussing technical issues, so that they can share their knowledge and solve problems with undocumented public libraries, unclear programming tasks, or new technologies or frameworks to explore [4]. Its design nurtures a community of developers, and enables crowdsourced software engineering activities ranging from documentation to production of useful, high quality code snippets [5]. The information available on this kind of social media service is also known as "crowd knowledge" and has the potential to become a substitute for the official software documentation [6]. Mamykina et al. conducted a statistical study on the entire SO corpus to find out what is behind of

*Correspondence to: {eduardocunha11, lucas.facom.ufu}@gmail.com, marcelo.maia@ufu.br

its immediate success [3]. Their findings showed that most of the questions will receive one or more answers (above 90% very quickly - with a median answer time of 11 minutes). Treude et al. pointed out that SO is particularly effective for code reviews, for conceptual questions and for novices [6].

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 search result is a set of discussions (threads), each one composed of a question and a series of answers. Users can sort threads according to a number of criteria such as: the number of votes the question received, or the relevance to the search query. However, considering only the textual similarity as the search criterion may be frustrating for the user. For example: the search can return threads to the user that despite having relevance to the query string, 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 intent. 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 programming tasks that developers are faced with. In this approach, we recommend pairs considering three aspects. The first one considers 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 questions 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 questions and answers to recommend question-answer pairs (i.e., Q&A pairs) that were well evaluated by the crowd. A Q&A pair is composed by a question and one answer for that question. For instance, if a SO's post has $n$ answers, there are $n$ possible Q&A pairs for that post, and each pair is composed by the question of SO's post and an answer for this question. The third aspect is related to question filtering: we aim at recommending only solutions to development tasks, which can be modeled as "how-to" posts that will be filtered by a classification algorithm. The result of the recommendation process is a ranked list of question-answer pairs.

In a previous conference paper [7], we conducted 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 (reproducible means that the snippet can be compiled and run inside the Integrated Development Environment (IDE) after minor adjustments or no adjustments). We have extended that paper in the following main points:

- We conducted a manual analysis of the top-10 recommendations produced by Google for all 35 development tasks considering the criteria *Relev* and *Reprod*, accounting for 350 new evaluation points for each criterion (700 in total);
- We have included a quantitative and qualitative comparison of our approach (*Lucene+Score+How-to*) with Google. The extended results are much more significant, because we improved the data analysis space and also because Google is currently the most popular general purpose search engine;
- We have detailed the *How-to* classifier used in the study, including new tables containing the information about the attributes of the classifier (see Tables II and III).

The rest of this paper is organized as follows. In Section 2, we present a Logistic Regression classifier used in this study to classify Q&A pairs into different categories. In Section 3, we present the description of our recommendation approach. In Section 4, we present the evaluation criteria and detail the experimental design of our work. In Section 5, we report the results that are discussed in Section 6. In Section 7 we present the Related Work, and finally in Section 8 we draw our conclusions.



DOI: 10.1002/smr

## 2. CLASSIFICATION OF Q&A PAIRS

On SO, users ask different kind of questions. Accordingly to Nasehi et al. [8] "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. [8] "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 SO". In this second dimension, they identified four categories of questions: *Debug-corrective*, *Need-to-know*, *How-to-do-it* and *Seeking-Different-Solution*. Similarly, we considered the following five categories in this second dimension:

- *How-to-do*: The questioner provides a scenario and a question about how to implement it (sometimes with a given technology or API) [8];
- *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 categories *Need-to-know* and *Seeking-Different-Solution* presented by Nasehi et al. [8] correspond respectively to our categories *Conceptual* and *Seeking-something* with some adjustments. The *How-to-do* category is very close to a scenario in which a developer has a programming task at hand and needs to solve it. For this reason, in our approach, we should filter Q&A pairs that are classified as *How-to-do*. In order to automate the selection of this kind of pairs, we developed a classifier to obtain only that type of Q&A pairs that is presented in the rest of this section.

### 2.1. Classification Algorithm

In order to choose the classification algorithm that best classifies Q&A pairs from SO, we performed a comparison between the following: Logistic Regression (LR) [9, 10], Naive Bayes (NB) [11], Multilayer Perceptron (MLP) [12], Support Vector Machine (SVM) [11], J4.8 Decision Tree (J4.8) [11, 13], Random Forest (RF) [14] and *K*-Nearest Neighbors (KNN) [11].

We decided to classify the Q&A pair instead of classifying only the question body because we observed that the answer body may provide relevant information to help to make the decision of the Q&A pair's category (e.g., distinguish between pairs of *How-to-do* and *Debug-corrective* categories).

### 2.2. SO Dataset

We downloaded a release of SO public data dump[†] (the version of March 2013) and imported the data into a relational database in order to classify the Q&A pairs. The table "posts" of this database

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




DOI: 10.1002/smr

stores all questions and all answers that were given to each question, if any, considering the date that the dump was built.

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 the five categories described above: *How-to-do*, *Conceptual*, *Seeking-something*, *Debug-corrective*, and *Miscellaneous*. We considered any Q&A pair belonging to any topic in order to avoid overfitting [15] or bias towards those specific topics. Table I shows the results of manual classification performed on 400 selected Q&A pairs. The two first authors of the paper performed a manual classification. First, each of the evaluators made an individual assessment of all selected pairs. Then, a consensual assessment was carried out to resolve conflicts and decide for the best classification. Of these 400 selected Q&A pairs, 74 (18.5%) had to be addressed in order to resolve the conflicts between evaluators. We identified that the major difficult in this classification process is to differentiate between categories *How-to-do* and *Debug-corrective*. The difference between these categories is subtle. Often both have code snippets in the question. When posting a question with code snippets in SO, the questioner may be wanting basically one of these two solutions: (i) a bug fix for the input code snippet (in the case of *Debug-corrective*) or (ii) a code example to perform a desired programming task (in the case of *How-to-do*).

Table I. Manual classification of 400 selected Q&A pairs.

| Category | #Classified Q&A pairs |
|---|---|
| *How-to-do* | 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 of the training dataset. Moreover, no practical application was found for *Miscellaneous* category. So, this category also was not considered in the construction of the training dataset.

In the next step, we generated an ARFF file (Attribute-Relation File Format), containing the labeled instances and the information of the classifier's attributes that was loaded into Weka [16]. We conducted an experimental study with 336 SO Q&A pairs divided into three domain categories: *How-to-do*, *Conceptual* and *Seeking-something*. The experiments were performed using a 10-fold-cross validation technique. The dataset used in this study is available online [‡].

### 2.3. Definition of Attributes

We defined 10 attributes to characterize SO Q&A pairs. Out of these 10 attributes, six are related to the number of occurrences of keyword terms in the "title", "question body" and "answer body" of a pair. The remaining four attributes are boolean ones 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. The considered keyword-based attributes are shown in Table II while the considered boolean attributes are shown in Table III.

The keywords used for classifying *How-to-do* questions shown in Table II are the result of a manual qualitative analysis of a sample of 100 SO posts. This sample was randomly selected and the first author of this paper identified the frequent words or expressions that appear in those posts. This analysis was performed in order to define the keyword-based attributes shown in Table II.

We decided not to use stemming in the keyword-based attributes of the classifier because the meaning of words is very important to assist the classifier in the decision process of the Q&A pair's category. Despite stemming has its advantages since this technique transforms different inflections and derivations of the same word to one common "stem", a problem concerning stemming is

[‡]http://lascam.facom.ufu.br/cms/ - "Menu Downloads / Paper Companions" (accessed and verified on 29/01/2016)



DOI: 10.1002/smr

Table II. Attributes and their respective keywords.

| Attribute | Keywords |
|---|---|
| howQty | **how to**, **how do**, **how does**, **how can**, **how i**, **how we**, **how you**, **way(s)**, method(s), function(s), algorithm(s), anyway, manner, mode, solution(s), pseudocode, script(s), workaround, solve, resolve, implement, step(s), approach, approaches |
| debugQty | exception(s), error(s), debug, debugging, fail, failed, warning, notice, notification, fault, problem, matter, trouble, wrong, incorrect, denied, breakpoint, unhandled, fix, bug(s), issue(s), tracker(s), permission(s), bug/feature |
| optimalQty | optimal, efficient, better, reliable, elegant, general, appropriate, suitable, adequate, proper, safest, fast, fastest, quickly, security, secure, robust, performant, performance, reasonably, smoother, viable, fast, lightweight, easy, easiest, cleanest, small, open-source, user-friendly, good, portable, correct, standard |
| lookingForQty | **tool(s)**, **tutorial(s)**, **manual(s)**, **book(s)**, **looking for**, **looking forward**, **looking at**, **looking around**, **searching for**, **searching forward**, **searching at**, **searching around**, client(s), find, app, application(s), lib(s), library, libraries, framework(s), migrate, migration(s), migrating, upgrade, convert, converting, conversion, porting, article(s), where, freeware, plugin(s), plug-in, research, search, seeking, google, system(s), video(s), resource(s), technique(s), editor(s), cms, erp, vmware, vpn, strategy, getting, started, when, ide(s), scanning, googling, blog(s), debugger(s), interpreter(s), compiler(s), comment(s), suggestion(s), looked, look, software(s), platform, profiler(s), generator(s), repository, repositories, should, advice(s), experience(s), experienced, used, replacement, idea(s), caveats, tips, tricks, recommend, recommendation, guideline(s), guide(s), guidance, orientation(s), help, helpful, suggest, suggestion(s), opinion(s), hint(s), point, pointers, experience(s), alternative(s), choice(s), thought(s), option(s), share, clue(s), insight, light, deal, dealt, package(s), available, threshold(s), freeware, direction(s), free, learning, material, beginner, possibilities, provider(s) |
| conceptualQty | **difference(s) between**, **is the**, **is this**, **are the**, **are this**, **best practice(s)**, **why**, **explain**, **clarify**, **explicate**, **explanation**, **explain**, meaning, significance, possible, what, what's, which, elucidate, illuminate, expound, tell, how much, how many, missing, level, metrics, statistics, reason, cause(s), justification(s), potential, concept, distinction(s), consensus, motive(s), mean, signify, signification, lesson(s), understand, explanatory, purpose(s), does, conceptual |
| questionQty | Returns the number of questions asked in the Q&A pair, based on the number of times that the operator mark (?) appears |

Table III. Definition of boolean attributes.

| Attribute | Definition |
|---|---|
| questionHasCode | Boolean value that indicates whether exists source code in the question. |
| answerHasCode | Boolean value that indicates whether exists source code in the answer. |
| questionHasLink | Boolean value that indicates whether exists link(s) in the question. |
| answerHasLink | Boolean value that indicates whether exists link(s) in the answer. |

the issue of overstemming. This problem occurs when unrelated words are conjoined under the same stem [17]. For instance, the words "general" (related to the "optimalQty" attribute) and "generator" (related to the "lookingForQty" attribute) are stemmed under the same stem "gener". Thus, the classifier will treat as the same, losing the notion of the meaning of these words. Although Porter stemmer [18] is known to be powerful, it still faces many problems. The major ones are overstemming and understemming errors [17].

In order to identify the frequent words or expressions that appear in those SO posts, we have built an *inverted index* (the first major concept in information retrieval). Each SO post has a unique serial number, known as the post identifier (*postID*). We keep a dictionary of terms (sometimes also referred to as vocabulary). We decided to consider the stop words in our vocabulary because they help to identify the main concerns of the questioners and what they wanted to solve in SO. Then for each term (word or expression), we have a list that records which SO posts the term occurs in (i.e.,



DOI: 10.1002/smr

for each term, we have a list of *postIDs*). Since a term generally occurs in a number of SO posts, this data organization already reduces the storage requirements of the index.

The dictionary also records some statistics, such as the number of SO posts which contain each term (the *post frequency* $p_f$, which is here also the length of each list). We defined that a term is frequent in our dictionary when $p_f \geq 30$ (i.e., the term appears at least in 30 of 100 SO posts considered). We tested with different values for $p_f$, but the accuracy of the How-to classifier was much lower. So, we decided to consider this threshold because we achieved better accuracy rate for the How-to classifier with this choice.

For each attribute related to keywords, we note the words and expressions that appeared more frequently in SO posts. We observed that the presence of source code in the question often occurs with Q&A pairs belonging to the *Debug-corrective* and *How-to-do* categories because the questioner posts the code snippet in the question in order to obtain a fix or solution for his/her problem. We also observed that often some answers of Q&A pairs belonging to the *How-to-do* and *Seeking-something* categories have links to external sources (e.g., tutorials, official API documentation). These links often have additional information about the programming task at hand. Thus, we consider the existence of the links in the experiment.

The bold keywords in Table II have weight five and the others have weight one. We adopted a weighting criterion because there are keywords that seem to be more important than others (i.e., some keywords help better in the decision process of the Q&A pair's category). We tried several different weights and we chose those that performed better (i.e., the weight in which the classifiers had the higher success rate). We conducted the experiments with the labeled Q&A pairs used to build the SO Dataset (more specifically the 336 labeled Q&A pairs divided into three domain categories: *How-to-do*, *Conceptual* and *Seeking-something*). We processed the question and answer texts of those Q&A pairs considering different weighting mechanisms for the relevant keywords, and for each mechanism, we generated a different ARFF file, containing the labeled instances and the information of attributes that was loaded into Weka [16]. The experiments were performed using a 10-fold-cross validation technique, considering the 5 most relevant attributes.

Table IV shows the classifier's accuracy (i.e., success rate) for different classifiers and weighting mechanisms experienced considering the 5 most relevant attributes. As shown in Table IV the higher success rates were achieved with weight five (i.e., column Weight = 5) for the words considered more relevant in the decision process of the Q&A pair's category (i.e., bold keywords in Table II). For all experienced weighting mechanisms, we considered that the weight for the less relevant keywords is equal to 1 (i.e., keywords that do not appear in bold in Table II).

Table IV. Classifier's Accuracy for different classifiers and weighting mechanisms considering the 5 most relevant attributes (Weight: the weight for words considered more relevant; ACC: the classifier's accuracy).

| Accuracy | Weight = 1 | Weight = 2 | Weight = 3 | Weight = 4 | Weight = 5 | Weight = 6 |
|---|---|---|---|---|---|---|
| ACC(LR) | 65.4223% | 66.6714% | 71.1315% | 74.2140% | 76.1905% | 75.2210% |
| ACC(NB) | 61.2917% | 62.3345% | 68.9127% | 69.8113% | 72.9167% | 70.2357% |
| ACC(MLP) | 64.2140% | 66.7132% | 71.2840% | 73.2546% | 75.8929% | 74.1018% |
| ACC(SVM) | 59.1439% | 59.7890% | 66.2453% | 68.4562% | 70.2381% | 69.4627% |
| ACC(J4.8) | 58.4567% | 58.9214% | 63.2718% | 67.3321% | 69.6429% | 68.3492% |
| ACC(RF) | 60.2143% | 62.7716% | 65.9122% | 68.1014% | 71.7262% | 70.2910% |
| ACC(KNN) (k = 5) | 58.8570% | 60.4310% | 63.2451% | 65.3160% | 69.9405% | 67.2314% |

Thus, each keyword has its own weight (one or five). Keywords with weight one are less important and are counted only once, while the keywords with weight five are considered more important and are counted five times when they appear in a Q&A pair. Moreover, we observed that keywords with weight five often appear in different Q&A pairs and describe clearly the concern of the questioner. In the next subsection, we detailed the feature selection process conducted in order to filter the most relevant attributes and we showed the classification results we obtained.



DOI: 10.1002/smr

### 2.4. *Feature Selection and Classification Results*

We carried out feature selection with information gain filter [19, 20] to reduce the feature space and eliminate the least relevant attributes. The filtering process selected six attributes: *conceptualQty, lookingForQty, howQty, answerHasCode, questionHasCode, answerHasLink*.

We performed information gain filter in the attributes in combination with ranker search method [§] (i.e., this method ranks attributes by their individual evaluations) considering a threshold = -1.7976931348623157E308. Table V shows the attributes in decreasing order of information gain value (the higher the value of information gain, the better the attribute contributes to the classification process).

Table V. Ranked Attributes: Information Gain

| Attribute | Information Gain Value |
|---|---|
| conceptualQty | 0.3309 |
| lookingForQty | 0.2486 |
| howQty | 0.2211 |
| answerHasCode | 0.0757 |
| questionHasCode | 0.0389 |
| answerHasLink | 0.029 |
| debugQty | 0 |
| questionHasLink | 0 |
| optimalQty | 0 |
| questionQty | 0 |

Table VI shows the classification results for 5 most relevant and 6 most relevant attributes selected by Information Gain Filter. We have obtained better results with the 5 most relevant attributes.

Table VI. Results with selected attributes: 5 most relevant and 6 most relevant.

| Classifier | top-5 attributes Success Rate | top-6 attributes Success Rate |
|---|---|---|
| LR | 76.1905% | 74.4048% |
| NB | 72.9167% | 70.8333% |
| MLP | 75.8929% | 72.3214% |
| SVM | 70.2381% | 70.5357% |
| J4.8 | 69.6429% | 69.6429% |
| RF | 71.7262% | 74.7024% |
| KNN (k = 5) | 69.9405% | 69.3452% |

The classification process can be performed at any current desktop computer running Weka software. It does not require fancy memory resources and processing. The largest cost is in the creation of the input file to the classifier (i.e., ARFF file) and not in the classification process itself. This input file contains information about the features of Q&A pairs belonging to a particular API. The whole classification process for a given API was executed in less than one hour.

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* category. We also obtained an overall success rate of 75.89% with a Multilayer Perceptron (MLP) classifier. In order to know whether LR and MLP were significantly different, we used a non-parametric test called Mc Nemar's test [21][22]. According to Mc Nemar's test, two algorithms can have 4 possible outcomes arranged in a 2x2 contingency table [23] as shown in Table VII.

$N_{ff}$ denotes the number of instances when both algorithms failed and $N_{ss}$ denotes success for both algorithms. These two cases do not give much information about the algorithm's performances as they do not indicate how their performances differ. However, the other two parameters ($N_{fs}$ and $N_{sf}$) show cases where one of the algorithms failed and the other succeeded indicating the

[§]http://weka.sourceforge.net/doc.dev/weka/attributeSelection/Ranker.html



DOI: 10.1002/smr

Table VII. Mc Nemar's test: Possible results of two algorithms.

| | Algorithm A failed | Algorithm A succeeded |
|---|---|---|
| Algorithm B failed | $N_{ff}$ | $N_{sf}$ |
| Algorithm B succeeded | $N_{fs}$ | $N_{ss}$ |

performance discrepancies. In order to quantify these differences Mc Nemar's test employs $v$ score (Equation 1) [22].

$$v = \frac{(|N_{sf} - N_{fs}| - 1)}{\sqrt{N_{sf} + N_{fs}}} \quad (1)$$

The $v$ score is interpreted as follows: When $v$ = 0, the two algorithms are said to show similar performance. As this value diverges from 0 in positive direction, this indicates that their performance differs significantly [22]. We used Mc Nemar's test for LR and MLP algorithms considering the best classification results we obtained (i.e., with 5 most relevant attributes, the overall success rates for LR and MLP were respectively, 76.1905% and 75.8929% in a total of 336 instances). Table VIII shows the 2x2 contingency table for LR and MLP classification algorithms. We obtained a $v$ score equal to 0. Thus, the classifiers LR and MLP were not significantly different. According to Mc Nemar's test, these classifiers have similar performance. Thus, in this case, we could choose anyone because it would not be a big difference if we adopt a classifier with a slightly lower success rate. In this study, we adopted the LR classifier to select Q&A pairs of the *How-to-do* category.

Table VIII. Mc Nemar's test: contingency table for LR and MLP algorithms ($v$ = 0).

| | Algorithm LR failed | Algorithm LR succeeded |
|---|---|---|
| Algorithm MLP failed | $N_{ff}$ = 80 | $N_{sf}$ = 1 |
| Algorithm MLP succeeded | $N_{fs}$ = 0 | $N_{ss}$ = 255 |

The final results of our recommendation strategy depend directly of a good classification. For instance, if a classifier has low success rate, it would misidentify a great number of Q&A pairs. It is essential to correctly classify *How-to* pairs because they have the characteristic of showing step by step how to solve a programming task at hand. The classification step is also important to improve the overall quality of the ranking. It would not be worth using another classifier in our study because so far we have not obtained a classifier better than LR. Only the MLP classifier achieved a similar performance (according to Mc Nemar's test). Therefore, the overall quality of the ranking generated by them would be very similar.

## 3. OUR APPROACH

In this section, we state our research goal, present the three topics used in the experiments and detail our recommendation approach.

### *3.1. Research Goal*

This paper aims at recommending Q&A pairs to assist developers in their development tasks, considering that the recommended Q&A pairs should have high textual similarity with the development task, they should be well evaluated by SO community, and they should be characterized as "How-to-do". A SO's post is formed by a question and a series of answers to that question. We decided to recommend Q&A pairs instead of entire posts because the answers for the same question can have different scores, i.e., some answers can be much better than others. We are interested in only recommend high quality content. We consider that a score of a pair (i.e., the number of upvotes minus the number of downvotes) is an indicative of its quality, because it represents the assessment of the crowd about the usefulness of the pair's content. So, we expect that recommended pairs are highly relevant in the context of the user task.



DOI: 10.1002/smr

### 3.2. Considered Topics

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

**Swing** is a toolkit to enable creating graphical user interfaces in Java. Developers can use Swing to create large-scale applications with a wide array of powerful components [24, 25].

**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 [26].

**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 [27].

### 3.3. Index Construction

We used the search engine Apache Lucene [28] to index the data. For a given topic (e.g., Swing) we obtain all threads from SO database in which the question has a specific tag (e.g., "swing"). Then, from that set of threads, we obtain all Q&A pairs, so, if a thread has a question and $n$ answers, we generate $n$ Q&A pairs for that thread. Table IX 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* pairs. Table X shows the result of the pairs' classification. For each Q&A pair classified as *How-to-do*, 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 [18]. As questions and answers from SO can have source code snippets that are not appropriate to be parsed using the Lucene's query parser (because it is primarily a natural language parser) we treat those snippets in a different way. For the Swing library we developed regular expressions to obtain the names of classes/interfaces/methods that are being created or called. 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 classic API: we checked if the source code snippet is using one of its operators (e.g., "OrderByDescending", "SelectMany", etc.). The regular expressions used for code term detection is available online [¶]. 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 XI 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 pairs classified in the category "How-to-do" because for each Q&A pair classified in that category a document is generated that composes the corpus used to build the index. Table XI also shows the number of different terms in the documents for each topic.

Table IX. Total of Q&A pairs by topic.

| Topic | Programming Language | Total of Q&A pairs |
|---|---|---|
| Boost | C++ | 14,558 |
| Swing | Java | 45,239 |
| LINQ | .NET Languages | 60,035 |

### 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 (list of terms) for that topic. We performed two types of search on this index, that we call *Scenario NKAE - Not Known API Element* and *Scenario KAE - Known API Element*.

[¶] http://lascam.facom.ufu.br/cms/ - "Menu Downloads / Paper Companions" (accessed and verified on 29/01/2016)



DOI: 10.1002/smr

Table X. Classification of Q&A pairs by topic.

| Topic | *How-to-do* | *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 XI. Index 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 |

*Scenario NKAE* 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 KAE* 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 API element has to be used.

We search Q&A pairs in *Scenario KAE* in a similar way of what we described for *Scenario NKAE*. 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 an operator name (in the case of LINQ) considered fundamental 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, one of the tasks selected for the experiment with Swing API has the title "Action Handling: Making Buttons Work". The class "ActionListener" is important in the solution of the task. Thus, we append the string "ActionListener" to the title, and the resulting string "Action Handling: Making Buttons Work ActionListener" is used as query string to search on Lucene's index (after removal of stop words and stemming).

The result of a search in Lucene index is a ranked list of documents (i.e., Q&A pairs), in which the first one is the most similar to the search query and the last one is the least 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 greatest *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 anything about its quality, i.e., among the pairs returned for a query, there may exist well evaluated and poorly evaluated pairs by SO community. Here we consider that a post's (question or answer) score is a proxy for its quality because the voting mechanism of 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 obtained better results with values 0.7 and 0.3 for the weights of the answer and question from a pair (In Subsection 4.3, we present the formal justification for consider these weights). Thus, we adopted these weights in our work. The reason to use this approach is that



DOI: 10.1002/smr

the answer seems to be more important than its belonging question, because it 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* 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 the 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 SO crowd. Both of these aspects are considered in the proposed recommendation, since we aim at providing pairs that are at the same time related to the problem and have good quality.

In order to combine both metrics into a single one, we performed a normalization step, because they have different nature. 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 its *SO score* in the range [0,1]. 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 as the search result.

## 4. EVALUATION PROCEDURES

In this section, we present our evaluation criteria and detail our experimental design.

### *4.1. Evaluation Criteria*

In this section, we present two criteria to evaluate each pair recommended by our approach.

The first criterion is called **Relevance** (in short, *Relev*). This criterion is used to assess to what extent the information contained in a pair can be used to help developers to solve the respective task. The grade given to this criterion ranges from 0 to 4. The value 0 means that the recommended pair is not related at all to the queried task. 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 what extent the source code snippets available in the question and answer bodies of a recommended pair can be easily compiled and executed. It is highly desirable that the code snippets could be run in isolation because developers frequently use source code examples as the basis for interacting with an API [29]. Thus, they need high quality examples of usage in order to learn how to use a desired API. One study found that the greatest obstacle to learning an API in practice is "insufficient or inadequate examples" [30]. Subramanian et al. [31] analyzed 39,000 code snippets given in response to SO questions and found that only 6,766 (17%) were complete files with class and method declarations, 6,302 (16%) code snippets were just method bodies devoid of class declarations, and the remaining 66% contained standalone source code statements (i.e., the majority are not compilable code fragments with complete class and method body declarations). Since the code is usually not complete, information present in the code is often not sufficient to resolve API method accesses. Moreover, they observed that most answers extend on details provided in the question; because of this, certain aspects of the snippet, like variable declarations are often skipped [31].

Concerning the *Reprod* criterion, we also evaluated code snippets that are present in external sources because several sites contain detailed information on how to solve a particular programming task. For example, if a page that the link represents has complete source code, the answer that includes the link is evaluated as "reproducible". While the criterion *Relev* has a semantic aspect, i.e., its main goal is to verify if the task can be solved using the recommended information, *Reprod* is a syntactic metric because it evaluates how easily the snippets can be compiled and executed,



DOI: 10.1002/smr

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 become complete and could be compiled).

### *4.2. Experimental Design*

We present experiments with the three considered topics to evaluate 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 [25], that contains only tasks related to GUI (Graphical User Interfaces) technologies. There are 14 tasks in that chapter and we randomly selected 12 of them.

The Boost tasks were extracted from a Boost Cookbook [26]. We randomly selected 12 of 91 tasks available in this cookbook.

The LINQ tasks were extracted from a blog[‖] 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 the other tasks described on the blog, and thus it is not appropriate to be used in an experiment to recommend pairs for LINQ, since it is much more related to a generic database field than to LINQ.

We conducted a manual analysis of the top-10 recommendations for all 35 tasks considering the criteria *Relev* and *Reprod*, accounting for 700 evaluation points. Moreover, we replicated the analysis using the Google search engine as baseline, accounting for more 700 evaluation points. The design of the Google analysis is presented in Subsection 4.4. Table XII shows the design of our experiment. For each topic (Swing, Boost and LINQ) we made experiments to test *Scenario NKAE* and *Scenario KAE*. From the 12 tasks previously selected for Swing, we randomly selected 6 for *Scenario NKAE* and 6 for *Scenario KAE*. The same was done for Boost. For LINQ, as we had only 11 tasks, we randomly selected 6 and 5 tasks for *Scenario NKAE* and *Scenario KAE* respectively. The input query for *Scenario NKAE* was the title of the tasks (after stemming and removal of stop words). The input for *Scenario KAE* 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 XIII shows for Swing, the 12 tasks selected for the experiment. The first 6 were included in *Scenario NKAE* and the remaining 6 in *Scenario KAE*. The task numbering in the first column corresponds to the numbering in the original document. The tasks for *Scenario KAE* 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 is 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 retrieve Q&A pairs. Similar tables are shown for Boost and LINQ (Tables XIV and XV respectively). In those tables, the name of classes or operators used for *Scenario KAE* are shown in italic.

The first two authors of this paper (let them be *Author A* and *Author B*) individually evaluated, for each of the 35 tasks, the top-10 recommended pairs. For each pair, they graded the two criteria previously described. In the Table XVI, the column "Kappa Before" shows the Weighted Kappa [32] 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

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



DOI: 10.1002/smr

Table XII. Experimental Design

| Topic | Scenarios | Tasks | Criteria | Ranker |
|---|---|---|---|---|
| Swing | Scenario NKAE | 6 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |
| | Scenario KAE | 6 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |
| Boost | Scenario NKAE | 6 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |
| | Scenario KAE | 6 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |
| LINQ | Scenario NKAE | 6 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |
| | Scenario KAE | 5 tasks | Relev | Google<br>Lucene+Score+How-to |
| | | | Reprod | Google<br>Lucene+Score+How-to |

Table XIII. Swing Tasks

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

Table XIV. Boost Tasks

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

for *Scenario NKAE* for Swing (i.e., that row represents a comparison between 60 values, since the authors analyzed the top-10 pairs recommended for each task).

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



DOI: 10.1002/smr

Table XV. LINQ Tasks

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

Table XVI. *Lucene+Score+How-to* Search: Weighted Kappa (Agreement Comparison).

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

(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 two is based on our interpretation that a difference of one could be considered a partial agreement among the evaluators instead of a disagreement.

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 XVI. 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*.

### *4.3. Evaluating different Weighting Mechanisms for a Q&A Pair*

In order to determine whether answers should have higher weight compared to questions, we analyzed different weighting mechanisms that increases answer's weight. In our study, $W_q$ and $W_a$ refer respectively to the weights for question and answer of a Q&A pair. We investigated the following weighting mechanisms:

- $W_q = 0.5$ and $W_a = 0.5$ (no weighting difference);
- $W_q = 0.3$ and $W_a = 0.7$;
- $W_q = 0.4$ and $W_a = 0.6$;
- $W_q = 0.2$ and $W_a = 0.8$;

In order to find out what weighting mechanism produces the best overall ranking we used a metric called *Normalized Discounted Cumulative Gain* (NDCG). This metric was used to have a numerical assessment of the ranking of pairs recommended in the experiments. NDCG (Equation 2) is generally used to evaluate retrieval results from search engines and uses a multi-valued notion of relevance [33]:



DOI: 10.1002/smr

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

where $k$ 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 $j$. Because 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 [34] to calculate this factor. In that approach, this factor is calculated in a scenario where all documents retrieved have the maximum grade (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 we use top-10 and they used top-15). However, using NDCG for this evaluation should be interpreted with caution, because only queries that had most of the 10 pairs with relevance close to four will have very high NDCG value, and that seems to be extremely stringent. We are mostly interested on the existence of 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.

To find the best weighting mechanism for our recommendation strategy (i.e., the one that produces the best overall ranking), we need to calculate the NDCG values for each criterion (i.e., *Relev* and *Reprod*) and for each weighting mechanism (i.e, $W_q$ = 0.5 and $W_a$ = 0.5, $W_q$ = 0.3 and $W_a$ = 0.7, $W_q$ = 0.4 and $W_a$ = 0.6, $W_q$ = 0.2 and $W_a$ = 0.8). Moreover, we need to compare these values with each other. Higher NDCG values imply better quality of the ranking.

A qualitative manual analysis of the top-10 recommendations for all 35 tasks considering the criteria *Relev* and *Reprod* is required to conduct the NDCG calculation. So, we start this analysis (shown above in Subsection 4.2) choosing a weighting mechanism, where any of them could be equally chosen because the order does not affect the result. The weighting mechanism $W_q$ = 0.3 and $W_a$ = 0.7 was arbitrarily selected and the ranking generated by it was considered as a comparison baseline with respect to the rankings generated by the remaining weighting options.

After this step, we calculated the ranking (i.e., top-10 recommendations for each development task queried in our system) generated by each weighting mechanism for the 35 development tasks listed above in Tables XIII, XIV and XV in order to identify which weighting mechanism is more suitable for our recommendation strategy (i.e., which weighting mechanism generates the best overall ranking). Each of these weighting mechanisms generates a different ranking. But we found that the difference between the rankings produced by different weighting mechanisms was very small (i.e., the top-10 recommended Q&A pairs were mostly repeated for different weighting mechanisms considering the same development task).

We note the new recommended Q&A pairs that were not present in the ranking generated by the weighting mechanism $W_q$ = 0.3 and $W_a$ = 0.7 (i.e., our comparison baseline) but were present in the ranking generated by the remaining weighting mechanisms (i.e., $W_q$ = 0.5 and $W_a$ = 0.5, $W_q$ = 0.4 and $W_a$ = 0.6 or $W_q$ = 0.2 and $W_a$ = 0.8). These new Q&A pairs were individually assessed by the first and third author of this paper (let them be *Author A* and *Author C*). For each of the 13 new recommended Q&A pairs, they graded the two criteria previously described. In the Table XVII, the column "Kappa Before" shows the Weighted Kappa [32] calculated to measure the agreement among the two evaluators. In that table, each row represents a criterion (*Relev* or *Reprod*). Thus, in the second row the Weighted Kappa was calculated to compare the *Reprod* grades given by the two authors for the 13 new recommended Q&A pairs.

Table XVII. New recommended Q&A pairs: Weighted Kappa (Agreement Comparison).

| Criterion | Kappa Before | Kappa After |
|---|---|---|
| *Relev* | 0.85 | 0.85 |
| *Reprod* | 0.56 | 0.89 |



DOI: 10.1002/smr

The evaluations of the two authors subsequently 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 *Reprod* as 0 for a pair and *Author C* graded *Reprod* as 2 for the same pair). The reason to consider only the differences greater than or equal to two is based on our interpretation that a difference of one could be considered a partial agreement among the evaluators instead of a disagreement.

After marking those pairs with major divergence, the evaluators discussed each one of the 5 divergent cases for *Reprod* criterion 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 XVII. 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). In the first row of Table XVII, the values for *Relev* criterion were the same because there was no difference greater than or equal to 2 between the grades of the evaluators in the 13 new recommended Q&A pairs. Since we have obtained high overall agreement, we decided to consider only the evaluations made by *Author A*.

Table XVIII shows the 13 new recommended Q&A pairs assessed by the two evaluators. For each Q&A pair, this table shows the Task Identifier, Topic, Scenario, Ranking Position in top-10 results (i.e., values ranging from 1 to 10), Weighting Mechanism and the final grades for *Relev* and *Reprod* criteria.

Table XVIII. New recommended Q&A pairs assessed by the two evaluators ($W_q$ and $W_a$ refer respectively to the weights for question and answer of a Q&A pair).

| Task | Topic | Scenario | Ranking Position | Weighting Mechanism | *Relev* | *Reprod* |
|---|---|---|---|---|---|---|
| 3.1 | Boost | NKAE | 10 | $W_q = 0.2$ $W_a = 0.8$ | 0 | 4 |
| 1.11 | Boost | KAE | 10 | $W_q = 0.2$ $W_a = 0.8$ | 0 | 1 |
| 9.2 | Boost | KAE | 8 | $W_q = 0.2$ $W_a = 0.8$ | 0 | 3 |
| 4 | LINQ | KAE | 10 | $W_q = 0.2$ $W_a = 0.8$ | 0 | 1 |
| 3.1 | Boost | NKAE | 8 | $W_q = 0.4$ $W_a = 0.6$ | 0 | 1 |
| 12.7 | Boost | NKAE | 9 | $W_q = 0.4$ $W_a = 0.6$ | 0 | 1 |
| 7.7 | Boost | KAE | 10 | $W_q = 0.4$ $W_a = 0.6$ | 0 | 3 |
| 9.2 | Boost | KAE | 10 | $W_q = 0.4$ $W_a = 0.6$ | 0 | 2 |
| 7.2 | Boost | KAE | 10 | $W_q = 0.4$ $W_a = 0.6$ | 4 | 3 |
| 12.7 | Boost | NKAE | 8 | $W_q = 0.5$ $W_a = 0.5$ | 0 | 1 |
| 12.7 | Boost | NKAE | 6 | $W_q = 0.5$ $W_a = 0.5$ | 0 | 1 |
| 10.7 | Boost | NKAE | 10 | $W_q = 0.5$ $W_a = 0.5$ | 0 | 1 |
| 3.5 | Boost | NKAE | 10 | $W_q = 0.5$ $W_a = 0.5$ | 0 | 4 |

After this evaluation step, for each weighting mechanism experienced, we calculated the $NDCG_{Relev}$ and $NDCG_{Reprod}$ for our approach (i.e., *Lucene+Score+How-to*). Table XIX shows the results we obtained.

Table XIX. $NDCG_{Relev}$ and $NDCG_{Reprod}$ values for each weighting mechanism experienced ($W_q$ and $W_a$ refer respectively to the weights for question and answer of a Q&A pair).

| | $W_q = 0.2$ $W_a = 0.8$ | $W_q = 0.4$ $W_a = 0.6$ | $W_q = 0.5$ $W_a = 0.5$ | $W_q = 0.3$ $W_a = 0.7$ |
|---|---|---|---|---|
| $NDCG_{Relev}$ | 0.3579 | 0.3565 | 0.3567 | 0.3583 |
| $NDCG_{Reprod}$ | 0.5249 | 0.5215 | 0.5228 | 0.5243 |

As we can see in Table XIX, there is no major difference in the NDCG values for the weighting mechanisms considered. The $NDCG_{Relev}$ for the weighting mechanism $W_q = 0.3$ and $W_a = 0.7$ was slightly better than the other ones. But the $NDCG_{Reprod}$ for the weighting mechanism $W_q = 0.2$ $W_a = 0.8$ was slightly better than the other ones. In this work, we present the results obtained with the weighting mechanism $W_q = 0.3$ and $W_a = 0.7$ (better value for $NDCG_{Relev}$). Nonetheless, the final results would not be very different if any of the other investigated weighting mechanisms were used.



DOI: 10.1002/smr

### 4.4. *Comparison of results with Google*

We compared our approach (*Lucene+Score+How-to*) to the Google web search. Google was chosen because it is the most popular general purpose search engine and it is generally used to query SO. We searched some tasks using SO's search engine. However, for several tasks, SO queries returned few posts. Thus, we observed that using the Google search engine to query SO was more effective than using SO's search engine itself. Again, *Author A* and *Author B* were the participants in this evaluation. We define a protocol for searching Google, based on the following rules:

- The search query must contain the API name and should be restricted to SO site (e.g., the search query "Program: Custom Font Chooser **Swing** site:stackoverflow.com" corresponds to a programming task using Java Swing API and is limited to SO site). We decided to do this to restrict our evaluation to only SO posts, because our criteria (*Relev* and *Reprod*) are more suitable to assess this type of content (e.g., it is difficult to evaluate if a book is relevant for a programming task at hand, given that the solution of the task is very specific);
- We set Google to search only posts that were posted on SO site until the date of our dump was performed, i.e., March 5, 2013. We decided to do this to ensure that the two approaches (*Lucene+Score+How-to* and Google) are dealing with the same data (i.e., the set of SO posts until March, 5, 2013).

We considered the same 35 development tasks evaluated in the previous step. For each task, we built the query string considering the above rule-based approach and searched in the Google site. *Author A* and *Author B* individually evaluated for each of the 35 tasks, the 10 first recommended SO posts considering the same criteria (*Relev* and *Reprod*). A SO post contains a question and one or more answers. The number of the answers for a question is the number of the Q&A pairs for that question. We decided to evaluate only the highest-scored Q&A pair that belongs to SO post, to have a fair comparison with our approach (Lucene+Score+How-to), which recommends Q&A pairs instead of whole SO posts. For each pair, *Author A* and *Author B* graded the two criteria previously described. We repeated the same process that was done for *Lucene+Score+How-to* approach (i.e., the evaluations of the authors were compared and the pairs with major divergence were discussed to get an agreement between the two evaluators). Table XX has the same column structure that XVI and shows the Weighted Kappa calculated to measure the agreement among the two evaluators. Again, since we have obtained high overall agreement, we decided to consider only the evaluations made by *Author A*.

Table XX. Google Search: Weighted Kappa (Agreement Comparison).

| Topic / Scenario / Criterion | Kappa Before | Kappa After |
|---|---|---|
| Swing / NKAE / *Relev* | 0.71 | 0.95 |
| Swing / NKAE / *Reprod* | 0.53 | 0.96 |
| Swing / KAE / *Relev* | 0.82 | 0.94 |
| Swing / KAE / *Reprod* | 0.56 | 0.96 |
| Boost / NKAE / *Relev* | 0.87 | 0.97 |
| Boost / NKAE / *Reprod* | 0.8 | 0.95 |
| Boost / KAE / *Relev* | 0.80 | 0.97 |
| Boost / KAE / *Reprod* | 0.65 | 0.97 |
| LINQ / NKAE / *Relev* | 0.88 | 0.99 |
| LINQ / NKAE / *Reprod* | 0.66 | 0.95 |
| LINQ / KAE / *Relev* | 0.81 | 0.98 |
| LINQ / KAE / *Reprod* | 0.479 | 0.83 |

## 5. RESULTS

In this section, we present the results of the proposed experiment. Tables XXI, XXII, XXIII, XXIV, XXV, XXVI, XXVII, XXVIII, XXIX, XXX, XXXI, XXXII have the same column



DOI: 10.1002/smr

structure: the task identifier and the first 10 ranking positions (P1 to P10) for each approach (i.e., *Lucene+Score+How-to* and Google). 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 (NKAE or KAE) and the criterion (*Relev* or *Reprod*).

For example in Table XXI, in the first line of data, we have the results of task 13.14 for both approaches (*Lucene+Score+How-to* and Google). In *Lucene+Score+How-to* approach, 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. In the same line, we can also observe the ranking obtained for Google approach in this development task. Furthermore, we can see highly relevant pairs at P1 and P7.

Table XXI. Swing - Scenario NKAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 13.14 | 2 | 2 | 4 | 2 | 2 | 4 | 2 | 2 | 2 | 2 | 4 | 2 | 2 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 13.13 | 0 | 4 | 4 | 4 | 3 | 3 | 1 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 3 | 2 | 0 | 3 | 4 | 4 |
| 13.11 | 1 | 2 | 2 | 0 | 2 | 2 | 3 | 0 | 1 | 2 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13.3 | 3 | 0 | 0 | 2 | 0 | 0 | 2 | 3 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 4 | 4 | 1 | 0 |
| 13.1 | 4 | 4 | 0 | 3 | 3 | 4 | 4 | 4 | 2 | 3 | 4 | 2 | 4 | 2 | 4 | 2 | 4 | 4 | 4 | 2 |
| 13.8 | 3 | 1 | 2 | 3 | 4 | 2 | 2 | 2 | 1 | 4 | 1 | 1 | 2 | 1 | 0 | 1 | 4 | 3 | 0 | 4 |

Table XXII. Swing - Scenario NKAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 13.14 | 0 | 0 | 4 | 3 | 0 | 3 | 4 | 3 | 2 | 0 | 4 | 3 | 3 | 3 | 4 | 4 | 4 | 1 | 2 | 0 |
| 13.13 | 3 | 0 | 0 | 4 | 4 | 0 | 0 | 0 | 4 | 3 | 3 | 4 | 4 | 3 | 0 | 0 | 0 | 0 | 4 | 4 |
| 13.11 | 0 | 0 | 0 | 3 | 3 | 3 | 4 | 4 | 3 | 0 | 0 | 0 | 3 | 2 | 2 | 4 | 4 | 4 | 2 | 2 |
| 13.3 | 0 | 4 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 4 | 4 | 1 | 0 |
| 13.1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 | 3 | 3 | 3 | 4 | 4 | 3 | 4 | 0 | 0 | 3 | 4 | 4 |
| 13.8 | 0 | 0 | 0 | 3 | 4 | 3 | 0 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 3 | 2 | 4 | 4 | 0 | 4 |

Table XXIII. Swing - Scenario KAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 13.12 | 4 | 2 | 3 | 3 | 4 | 0 | 2 | 0 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 0 | 3 | 4 | 1 | 4 |
| 13.2 | 3 | 1 | 2 | 1 | 4 | 4 | 1 | 0 | 0 | 2 | 3 | 4 | 4 | 4 | 4 | 2 | 4 | 1 | 2 | 0 |
| 13.9 | 4 | 4 | 4 | 1 | 1 | 0 | 0 | 4 | 0 | 0 | 4 | 4 | 0 | 4 | 4 | 0 | 0 | 1 | 3 | 0 |
| 13.4 | 4 | 2 | 2 | 2 | 3 | 2 | 1 | 3 | 3 | 3 | 4 | 0 | 3 | 0 | 1 | 4 | 2 | 0 | 4 | 4 |
| 13.5 | 4 | 2 | 4 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 0 | 4 | 3 | 4 | 3 | 3 | 4 | 1 |
| 13.6 | 2 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 3 | 2 | 4 | 4 | 4 |

Table XXIV. Swing - Scenario KAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 13.12 | 0 | 0 | 0 | 4 | 3 | 3 | 4 | 0 | 2 | 0 | 0 | 4 | 4 | 4 | 3 | 3 | 4 | 4 | 4 | 4 |
| 13.2 | 4 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 2 | 2 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 |
| 13.9 | 4 | 4 | 4 | 3 | 4 | 4 | 4 | 4 | 0 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 |
| 13.4 | 4 | 0 | 0 | 3 | 4 | 4 | 0 | 4 | 4 | 4 | 4 | 0 | 4 | 3 | 0 | 4 | 2 | 0 | 4 | 4 |
| 13.5 | 0 | 4 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 4 | 4 | 2 |
| 13.6 | 3 | 4 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 1 | 4 | 3 | 3 | 4 | 4 | 4 | 3 |

We calculated the NDCG$_{Relev}$ and NDCG$_{Reprod}$ values for each approach. Table XXXIII shows the obtained results. We can state that NDCG$_{Relev}$ for Google was 5.62% higher than *Lucene+Score+How-to* approach and NDCG$_{Reprod}$ for Google was 4.75% higher than *Lucene+Score+How-to* approach. These results show that the overall quality of the recommended pairs by Google was slightly better than the *Lucene+Score+How-to* approach.



DOI: 10.1002/smr

Table XXV. Boost - Scenario NKAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 2.8 | 3 | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 2 | 3 | 4 | 4 |
| 3.1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 12.5 | 1 | 0 | 0 | 0 | 4 | 0 | 4 | 4 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12.7 | 2 | 4 | 0 | 0 | 0 | 2 | 2 | 2 | 4 | 0 | 2 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10.7 | 3 | 2 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3.5 | 3 | 2 | 2 | 2 | 3 | 2 | 2 | 3 | 2 | 3 | 3 | 3 | 1 | 3 | 3 | 3 | 3 | 4 | 3 | 0 |

Table XXVI. Boost - Scenario NKAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 2.8 | 2 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 2 | 4 | 4 | 4 |
| 3.1 | 4 | 4 | 4 | 0 | 4 | 1 | 0 | 2 | 0 | 0 | 4 | 3 | 0 | 4 | 3 | 3 | 3 | 2 | 3 | 3 |
| 12.5 | 0 | 0 | 0 | 0 | 4 | 0 | 4 | 4 | 0 | 0 | 3 | 3 | 0 | 0 | 0 | 1 | 3 | 3 | 0 | 0 |
| 12.7 | 0 | 0 | 0 | 3 | 4 | 0 | 0 | 4 | 3 | 4 | 1 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10.7 | 4 | 0 | 4 | 0 | 4 | 0 | 3 | 4 | 4 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3.5 | 4 | 0 | 4 | 0 | 0 | 4 | 0 | 0 | 0 | 4 | 0 | 3 | 4 | 0 | 0 | 3 | 3 | 0 | 0 | 3 |

Table XXVII. Boost - Scenario KAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 7.7 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10.2 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1.11 | 3 | 1 | 3 | 1 | 1 | 3 | 3 | 2 | 2 | 1 | 4 | 2 | 0 | 3 | 4 | 0 | 4 | 0 | 0 | 0 |
| 9.2 | 2 | 1 | 1 | 1 | 3 | 1 | 3 | 1 | 1 | 0 | 2 | 0 | 0 | 3 | 2 | 2 | 4 | 3 | 4 | 0 |
| 7.2 | 4 | 4 | 4 | 4 | 3 | 4 | 0 | 2 | 2 | 4 | 4 | 4 | 4 | 0 | 0 | 4 | 4 | 4 | 2 | 4 |
| 8.8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table XXVIII. Boost - Scenario KAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 7.7 | 4 | 0 | 4 | 1 | 4 | 2 | 4 | 4 | 4 | 4 | 4 | 0 | 4 | 3 | 0 | 4 | 4 | 4 | 3 | 1 |
| 10.2 | 4 | 0 | 0 | 2 | 4 | 4 | 0 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1.11 | 0 | 3 | 0 | 4 | 2 | 0 | 4 | 4 | 4 | 2 | 4 | 4 | 0 | 2 | 4 | 4 | 3 | 0 | 2 | 1 |
| 9.2 | 0 | 4 | 0 | 3 | 4 | 4 | 3 | 4 | 2 | 2 | 0 | 2 | 4 | 3 | 4 | 0 | 1 | 4 | 4 | 2 |
| 7.2 | 0 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 4 | 4 | 4 | 4 |
| 8.8 | 4 | 4 | 4 | 4 | 2 | 4 | 3 | 3 | 3 | 4 | 1 | 0 | 3 | 2 | 2 | 4 | 3 | 4 | 3 | 0 |

Table XXIX. LINQ - Scenario NKAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 2 | 0 | 0 | 0 | 2 | 2 | 0 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 2 | 4 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 2 | 4 | 2 | 0 | 0 | 4 | 4 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 2 | 0 | 2 | 1 | 2 | 2 | 2 | 2 | 0 | 2 | 0 | 3 | 3 | 4 | 3 | 0 | 0 | 0 | 0 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | 4 | 4 | 4 | 4 | 1 | 4 | 3 | 1 | 3 | 0 | 4 | 4 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |

Table XXX. LINQ - Scenario NKAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 2 | 3 | 0 | 3 | 4 | 4 | 4 | 4 | 0 | 3 | 3 | 4 | 3 | 0 | 4 | 4 | 4 | 1 | 1 | 0 | 3 |
| 10 | 0 | 0 | 3 | 3 | 2 | 0 | 3 | 0 | 0 | 0 | 4 | 3 | 0 | 4 | 2 | 0 | 0 | 3 | 3 | 0 |
| 1 | 4 | 1 | 0 | 0 | 2 | 2 | 2 | 4 | 4 | 0 | 4 | 4 | 4 | 2 | 4 | 4 | 0 | 4 | 0 | 1 |
| 5 | 3 | 2 | 4 | 0 | 3 | 4 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 3 | 3 | 3 |
| 11 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 0 | 3 | 0 | 4 |
| 12 | 3 | 3 | 3 | 3 | 0 | 3 | 4 | 2 | 4 | 0 | 3 | 3 | 4 | 4 | 4 | 0 | 3 | 3 | 4 | 4 |



DOI: 10.1002/smr

Table XXXI. LINQ - Scenario KAE (0 = Not Relevant, 4 = Highly Relevant).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 3 | 1 | 1 | 2 | 1 | 4 | 1 | 0 | 2 | 1 | 1 | 4 | 0 | 0 | 2 | 4 | 0 | 0 | 0 | 0 | 0 |
| 7 | 4 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 2 | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 2 | 4 | 4 | 2 | 2 | 4 | 4 | 1 | 4 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 0 | 4 | 4 | 4 |
| 8 | 0 | 2 | 1 | 2 | 1 | 0 | 0 | 3 | 4 | 3 | 4 | 4 | 2 | 4 | 4 | 4 | 2 | 4 | 2 | 2 |

Table XXXII. LINQ - Scenario KAE (0 = Not Reproducible, 4 = Highly Reproducible).

| | *Lucene+Score+How-to* | | | | | | | | | | Google | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 |
| 3 | 4 | 4 | 4 | 4 | 3 | 3 | 3 | 4 | 4 | 2 | 3 | 3 | 3 | 4 | 4 | 3 | 4 | 4 | 3 | 4 |
| 7 | 3 | 4 | 4 | 3 | 2 | 3 | 3 | 2 | 4 | 3 | 4 | 4 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 3 | 0 | 3 | 3 | 4 | 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 3 | 0 | 2 | 4 | 0 | 0 | 4 |
| 9 | 3 | 4 | 4 | 3 | 3 | 4 | 3 | 2 | 3 | 2 | 4 | 4 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 8 | 3 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 4 | 0 | 3 | 3 | 4 | 3 | 3 | 4 | 4 | 3 | 4 | 4 |

Table XXXIII. $\text{NDCG}_{Relev}$ and $\text{NDCG}_{Reprod}$ values for each approach.

| | *Lucene+Score+How-to* | Google |
|---|---|---|
| $\text{NDCG}_{Relev}$ | 0.3583 | 0.4145 |
| $\text{NDCG}_{Reprod}$ | 0.5243 | 0.5718 |

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 *Lucene+Score+How-to* approach and by Google search engine having *Relev* or *Reprod* greater than or equal to 3. For this analysis, we use the graphics shown in Figures 1 and 2. These figures show the scatter plots for criteria *Relev* and *Reprod* of each recommendation approach. The three graphics in the first column of the Figure 1 consider the criterion *Relev* for *Lucene+Score+How-to* in the three topics assessed in this study (Boost, LINQ and Swing), while the remaining three graphics in the second column of this figure consider the same criterion for Google approach. Figure 2 is similar to Figure 1, but takes into account the criterion *Reprod* instead of criterion *Relev*.

Each graphic considers the selected activities for *Scenario NKAE* (diamond symbol) and the ones for *Scenario KAE* (square symbol). The activities appear in the graphics in the order shown in Tables XIII, XIV and XV. Thus, in the first line of the Figure 1 (i.e., the graphics "*Lucene+Score+How-to*: Boost - *Relev*" and "Google: Boost - *Relev*"), the *activity 1* of each scenario corresponds to "2.8 - Parsing date-time input" (*Scenario NKAE*) and "7.7 Using a reference to string type *string_ref*" (*Scenario KAE*), as can be observed in Table XIV. 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 solve almost 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 of Figure 1 for *Lucene+Score+How-to* approach, all activities for Swing had at least one pair with *Relev* $\geq 3$. Only eight of the 35 tasks (22.85%) had no pairs with *Relev* $\geq 3$ among the 10 recommended pairs. Concerning the graphics of this same figure for Google, we can see that all activities for Swing had at least one pair with *Relev* $\geq 3$. Out of the 35 tasks, 10 (28.57%) had no pairs with *Relev* $\geq 3$ among the 10 recommended pairs, which is a worse performance compared to the *Lucene+Score+How-to* approach.

As we can see in the graphics of Figure 2 for *Lucene+Score+How-to* approach, all the 35 activities had at least one pair with *Reprod* $\geq 3$. Concerning the graphics of this same figure for Google, we can see that one out of the 35 tasks (2.85%) had no pairs with *Reprod* $\geq 3$. This activity corresponds to "10.7 - The portable way to export and import functions and classes" (Boost - *Scenario NKAE*). For this activity, Google returned some SO posts related to the Python language. These SO posts contained nothing about library Boost of C++ language. This type of strange behavior on the Google's recommendation is not desirable and we observed that this behavior is



DOI: 10.1002/smr

not an isolated case. To demonstrate this point, we conducted a qualitative analysis of all tasks in which Google and *Lucene+Score+How-to* produced a bad recommendation.

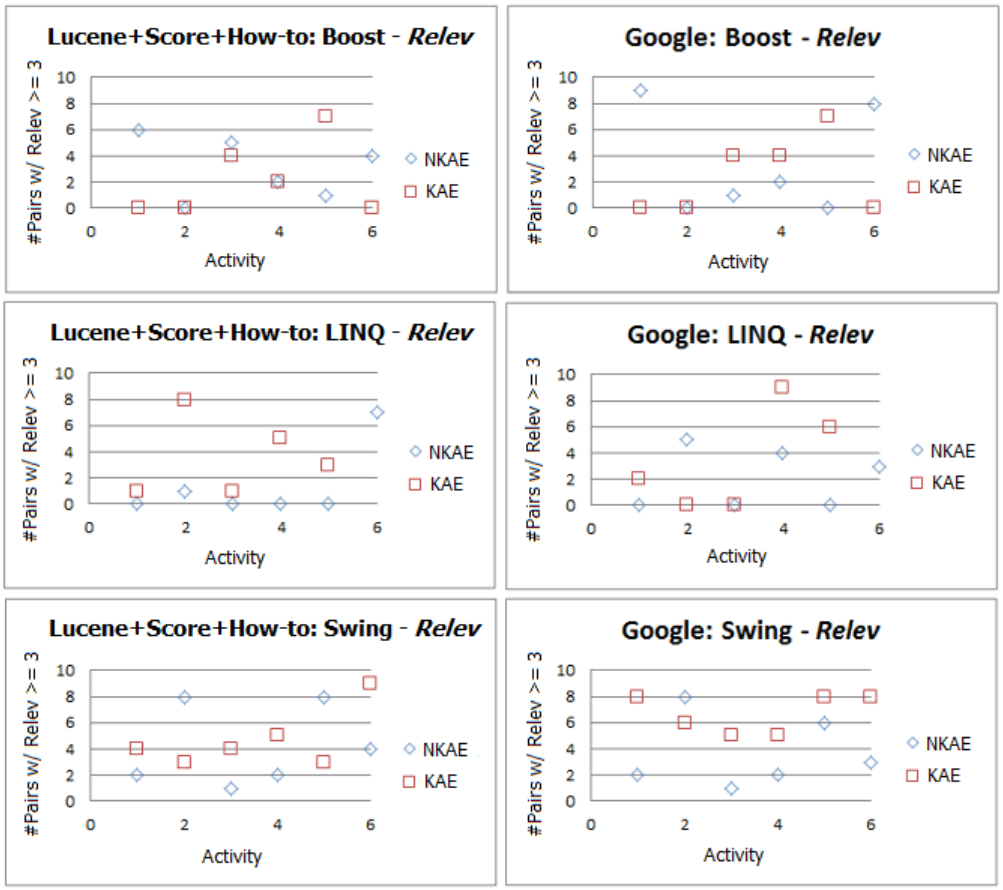


Figure 1. Numbers of pairs having *Relev* $\geq 3$ (*Lucene+Score+How-to*: Left, Google: Right).

In order to understand exactly what kind of queries Google can deal with better than *Lucene+Score+How-to* approach, and vice versa, we deepen our qualitative analysis. For each case (i.e., "Task / Topic / Scenario / Criterion"), we calculate the amount of grades 3 and 4 (i.e., good values for *Relev* and *Reprod* criteria). If this amount is greater than or equal to 6, we assume it is a case with good recommendation. Otherwise, we assume it is a case with bad recommendation. Moreover, for each case, we analyzed the quality of the recommendations made by each approach.

Using this assumption, we selected the cases in which *Lucene+Score+How-to* outperformed Google, and vice versa. Tables XXXIV and XXXV show these cases. We outperformed Google in 8 cases and Google outperformed *Lucene+Score+How-to* in 14 cases.

Table XXXIV shows the cases in which *Lucene+Score+How-to* approach outperformed Google. Of 8 cases that *Lucene+Score+How-to* can deal better than Google, 5 (62.5%) were belonging to the scenario NKAE (*Not Known API Element*). These results suggest that our approach copes well with generic queries that do not have the API element. However, Google produces a bad recommendation with this kind of query. We observed that in these cases, Google recommends SO's posts that were not tagged with the desired API. Thus, our approach tends to be more effective than Google in the Scenario NKAE. Furthermore, 4 out of 8 cases (50%) were belonging to the Boost topic. These results show that *Lucene+Score+How-to* was better than Google on the Boost topic. Table XXXVI shows some topics that Google recommended that were not related with Boost and LINQ APIs. In 80% of these cases shown in Table XXXVI, the Scenario was NKAE.



DOI: 10.1002/smr

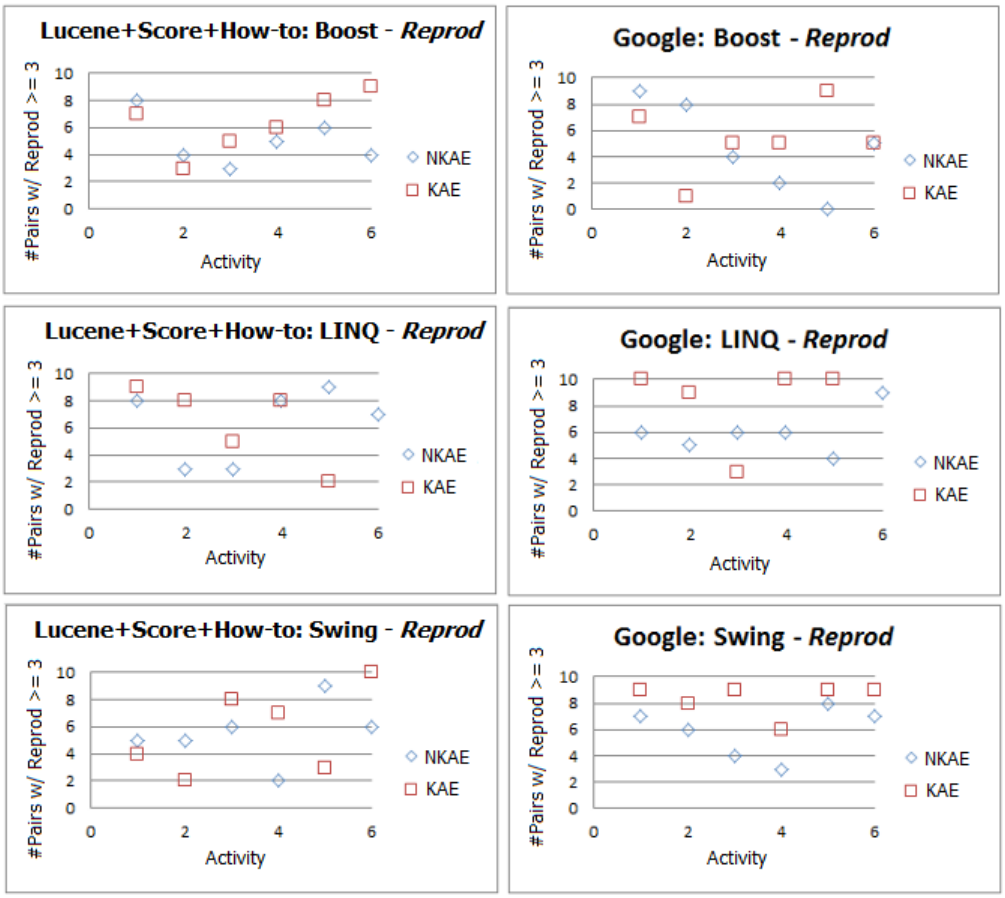


Figure 2. Numbers of pairs having *Reprod* $\geq 3$ (*Lucene+Score+How-to*: Left, Google: Right).

Table XXXV shows the cases in which Google outperformed *Lucene+Score+How-to* approach. Considering those 14 cases, 9 were related to the Scenario KAE (*Known API Element*), being 6 of Swing API and 3 of LINQ API. These results show that Google tends to be more effective than our approach (i.e., using regular expressions to obtain the names of API elements from code snippets) in the Scenario KAE.

Our assumption is that if the API is quite popular (i.e., have a high number of tagged SO posts like Swing) in SO and Scenario is KAE, Google tends to perform better than our approach because of its sophisticated probabilistic learning mechanism. This could explain why Google was better on Swing topic, since this API is quite popular in SO and 6 out of 8 Swing cases shown in Table XXXV were related to the Scenario KAE. The result suggest that our approach tends to perform better in smaller than larger indexes of documents because our filtering mechanisms are still useful and Google would have less learning opportunity. If this assumption is true, our approach tends to be better for less popular APIs in SO (e.g., Boost, Log4j, JFreeChart) than for most popular APIs in SO (e.g., Swing, Hibernate, Spring). This could explain why our approach outperformed Google on Boost topic.

For LINQ topic (despite being a popular topic in SO), neither approach performed very well. Maybe SO does not contain solutions for these tasks. Moreover, some LINQ tasks do not have titles sufficiently specific (e.g., the task's title *"Desktop Search Statistics"* is very generic).

In order to confirm this assumption, it would be necessary to perform another qualitative assessment involving the new APIs and the new Q&A recommended pairs that would require a significant effort and unfortunately is left as future work.



DOI: 10.1002/smr

Table XXXIV. Cases in which *Lucene+Score+How-to* outperformed Google.

| Task | Topic / Scenario / Criterion | Task's Title |
|---|---|---|
| 13.11 | Swing / NKAE / *Reprod* | Choosing a Color from all the colors available on your computer |
| 12.5 | Boost / NKAE / *Relev* | Using portable math functions |
| 10.7 | Boost / NKAE / *Reprod* | The portable way to export and import functions and classes |
| 9.2 | Boost / KAE / *Reprod* | Using an unordered set and map *unordered_set* |
| 8.8 | Boost / KAE / *Reprod* | Splitting a single tuple into two tuples |
| 12 | LINQ / NKAE / *Relev* | Calculate the Standard Deviation |
| 11 | LINQ / NKAE / *Reprod* | Desktop Search Statistics |
| 7 | LINQ / KAE / *Relev* | Selecting Pages of Data from Northwind *Skip* |
| | | **Total: 8** |

Table XXXV. Cases in which Google outperformed *Lucene+Score+How-to*.

| Task | Topic / Scenario / Criterion | Task's Title |
|---|---|---|
| 13.14 | Swing / NKAE / *Reprod* | Program: Custom Font Chooser |
| 13.13 | Swing / NKAE / *Reprod* | Changing a Swing Program's Look and Feel |
| 13.12 | Swing / KAE / *Relev* | Centering a Main Window *JFrame* |
| 13.2 | Swing / KAE / *Relev* | Adding and Displaying GUI Components to a window *JFrame* |
| 13.5 | Swing / KAE / *Relev* | Action Handling: Making Buttons Work *ActionListener* |
| 13.12 | Swing / KAE / *Reprod* | Centering a Main Window *JFrame* |
| 13.2 | Swing / KAE / *Reprod* | Adding and Displaying GUI Components to a window *JFrame* |
| 13.5 | Swing / KAE / *Reprod* | Action Handling: Making Buttons Work *ActionListener* |
| 3.5 | Boost / NKAE / *Relev* | Reference counting pointers to arrays used across methods |
| 3.1 | Boost / NKAE / *Reprod* | Doing something at scope exit |
| 1 | LINQ / NKAE / *Reprod* | Change the font for all labels on a windows form |
| 9 | LINQ / KAE / *Relev* | Dynamic Sort Order *OrderByDescending* |
| 8 | LINQ / KAE / *Relev* | Querying XML Using LINQ *Contains* |
| 8 | LINQ / KAE / *Reprod* | Querying XML Using LINQ *Contains* |
| | | **Total: 14** |

Table XXXVI. Noise cases in Google's recommendation.

| Task | Topic / Scenario / Criterion | Some topics that Google recommended |
|---|---|---|
| 12.5 | Boost / NKAE / *Relev* | SIMD library, C language |
| 10.7 | Boost / NKAE / *Relev* | Python language, XML |
| 8.8 | Boost / KAE / *Reprod* | Scala language, Python language, SQLite |
| 11 | LINQ / NKAE / *Reprod* | SqlServer, Postgresql, Twitter |
| 12 | LINQ / NKAE / *Relev* | SQLite |

## 6. DISCUSSION

We could partially compare our results with the results presented by Ponzanelli et al. [34] 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 three aspects to suggest content: the textual similarity that the pairs have with the search query, their score, and their "How-to" nature. In their approach, only the textual relevance is considered. We consider this *How-to* 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);
- 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



DOI: 10.1002/smr

programming course, we focus more on practical tasks that a developer can face daily than on didactic items used to teach a topic;
- In their paper, they recommended entire SO threads. Here, we recommend individual Q&A pairs, because a question can receive 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 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).

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. Some caution is necessary to interpret those numbers because SEAHAWK recommends 15 Q&A threads, and in this work, we analyzed 10 Q&A pairs.

Analyzing Figure 1, we can see that for criterion *Relev*, Swing has shown better results over Boost and LINQ since for all tasks it had at least one pair with *Relev* $\geq 3$ for both recommendation approaches (i.e., *Lucene+Score+How-to* and Google) and *Scenarios* (i.e., *NKAE* and *KAE*).

Considering Figure 2, we can state that for criterion *Reprod*, both recommendation approaches have good results in the recommendation of reproducible Q&A pairs, since for *Lucene+Score+How-to* approach, all the 35 tasks had at least one pair with *Reprod* $\geq 3$ and for Google, only one of the 35 tasks (2.85%) had no pairs with *Reprod* $\geq 3$.

There could be one main reason to explain the low number of pairs with *Relev* $\geq 3$ for some tasks in our approach (*Lucene+Score+How-to*). First, this approach uses the title of a task as an input in the search engine. Some tasks do not have a precise information on its title. For instance, the task #5 "11. 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 titles sufficiently specific. In this case, the result was equally poor using the Google search.

All 35 tasks had at least one recommended pair with *Reprod* $\geq 3$ 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 explain the 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 creating 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.

We can observe from Figures 1 and 2, that 26 out of the 35 tasks of our approach have at least one recommended Q&A pair which has both criteria *Relev*, *Reprod* $\geq 3$.

Despite $\text{NDCG}_{Relev}$ and $\text{NDCG}_{Reprod}$ values were lower in the *Lucene+Score+How-to* than Google (see Table XXXIII), we identified some weaknesses in Google's recommendation:

- Firstly, the number of poor results (i.e., Q&A pairs with grade 0 in some of the criteria *Relev* or *Reprod*) for Google was higher in Boost and LINQ topics than *Lucene+Score+How-to* approach. The second and third line of data of Table XXXVII shows that Google obtained 46.66% and 40.90% of poor results for Boost and LINQ topics, respectively, whereas *Lucene+Score+How-to* obtained 32.08% and 31.81% of poor results to the topics listed in the same order. Furthermore, the number of best results (i.e., Q&A pairs with grade 4 in some



DOI: 10.1002/smr

of the criteria *Relev* or *Reprod*) for *Lucene+Score+How-to* was higher in Boost topic than Google approach. The second line of data of Table XXXVII shows that *Lucene+Score+How-to* approach obtained 30% of best results for Boost topic, while Google obtained only 25% in this topic;

- Secondly, some SO posts recommended by Google were not related to the respective API (e.g., for the task 10.7 of Boost called "The portable way to export and import functions and classes Boost", Google returned some SO posts related to the Python language. These SO posts contained nothing about library Boost of C++ language. There are other development tasks in which Google returns SO posts that were not related to the respective API, like the tasks 11 and 12 of LINQ - Scenario NKAE). This unexpected behavior called us attention because we defined that the search query must contain the API name, and Google returned results that were not related to the respective API. Our approach does not suffer from this problem because we filtered only SO posts related with the API of interest using the tag field;
- Thirdly, some SO posts recommended by Google were more theoretical than practical (e.g., for the task 12.7 of Boost "Combining multiple test cases in one test module", Google returned some theoretical SO posts that explain what is an unit test, a regression test, etc. instead of an usage example using the library Boost of C++ language). We suggest that the absence of a *How-to* classifier had contributed to this undesired scenario for Google.

We also identified some weaknesses in our approach:

- Firstly, we could have produced a customized binary classifier because the only distinction that matters for this work is "how-to-do" vs. "not how-to-do" question-answer pairs. Arguably, a classifier trained on just these two categories could achieve better results;
- Secondly, the choice of tags to detect topics seems to be at least a bit potentially problematic. The tags are provided manually by SO participants and are therefore known to be unreliable (e.g., due to synonymy and polysemy problems, despite the best efforts of SO community). This realisation has lead to repeated attempts to predict the tags based on the question text (e.g., by Wang et al. [35]).

These results suggest that the construction of a hybrid approach using the Google search engine to match textual and semantic similarity enhanced with a *How-to* classifier would provide the best possible results considering the presented alternatives.

Table XXXVII. Quantity of Poor and Best Results per approach.

| | Qty. Poor Results (Grade = 0) | | Qty. Best Results (Grade = 4) | |
|---|---|---|---|---|
| Topic | *Lucene+Score+How-to* | Google | *Lucene+Score+How-to* | Google |
| Swing | 27.91% | 21.66% | 30.83% | 48.75% |
| Boost | 32.08% | 46.66% | 30% | 25% |
| LINQ | 31.81% | 40.90% | 24.09% | 27.27% |

### *6.1. Threats to Validity*

Some threats should be considered in the analysis of the presented results.

The choice of tags to detect topics seems to be at least a bit potentially problematic. If the tag is not correct (e.g., the tag was misspelled by the tool user), our approach will not be able to retrieve the threads from SO database in which the question has this incorrect tag. Thus, no Q&A pair will be recommended to the end user of the proposed approach. Tags assigned to SO questions tend to be noisy and some SO questions are not well tagged [35]. Most software information sites allow users to create tags freely. However, this freedom comes at a cost, as tags can be idiosyncratic due to users' personal terminology [35] [36]. Furthermore, some software information sites (e.g., SO) require users to add at least 3 tags at the time of posting a question, even if they are unfamiliar with the tags in circulation at that time. As tagging is inherently a distributed and uncoordinated process, often similar questions are tagged differently [37]. For example, in SO the tags *xmlparser*,



DOI: 10.1002/smr

*xml-parser* and *xmlparsing* are all used to describe a parser of an XML file. Idiosyncrasy reduces the usefulness of tags, since related objects are not linked together by a common tag and relevant information becomes more difficult to retrieve [35]. Another phenomenon in software information sites is "Tag Synonyms". This phenomenon refers to tags which are syntactically different (i.e., they are different strings of symbols) but are semantically the same (i.e., they have the same meaning) [37].

One way the website can facilitate accurate tagging is to recommend tags for users based on the content they generate. The development of a tag recommendation system for user created content is a relatively new field. Thus, tag recommendation is a field in which the state of the art is still being actively developed, and the most accurate methods for recommending tags have yet to be established. We discovered two recent papers that detail two algorithms, *TagCombine* [37] and *EnTagRec* [35] for tag recommendation on sites like SO. These algorithms use machine learning for tag recommendation.

Considering the construct validity, the rating process of SO Q&A pairs is human-dependent. Therefore, the disagreement among people about the grade of a Q&A pair is possible. In order to mitigate this threat, we conducted a two-round evaluation process to improve the weighted-Kappa agreement of evaluators. Another threat in this category raises when selecting a class to compose queries for tasks in *Scenario KAE*. There could be more than one class in a snippet and different choices could lead to different results. This threat is minimized because even when there are more than one class in the snippet, the chosen class is generally quite obvious because snippets tend to be small, thus facilitating the identification of the dominant class.

We evaluated a Q&A pair knowing whether it came from Google or our own approach (i.e., *Lucene+Score+How-to*). This is a threat to validity because ideally the assessment should have been blind, i.e., without knowing which approach recommended the Q&A pair.

Concerning conclusion validity, we can identify a threat on the definition of measures for what means a useful pair in the context of a programming activity. Even, if the semantic adherence of the post (*Relev*) and its reproducibility (*Reprod*) are meaningful representatives of the usefulness of the posts, the adoption of other measures could refine the evaluation of posts.

At last, but not least, concerning the external validity, our results have been conducted with three different APIs and we could observe a variability on the results depending on the API. There are several factors that could affect the results for a specific API. For instance, if the coverage of an API is not appropriate or even if the API does not attract the interest of the community, the results are not expected to be promising as shown in this work. Moreover, for each API, the chosen tasks may not represent the typical need of developers during software development. However, our option to select those tasks from third-party documents was an important achievement because we could eliminate the bias of arbitrary choices.

## 7. RELATED WORK

Treude et al. [6] 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's 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.

Nasehi et al. [8] showed that 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. The second dimension is about the main concerns of the questioners and what they wanted to solve. They identified four types in this last dimension: *Debug/Corrective*, *Need-to-Know*, *How-To-Do-It*, and *Seeking-Different-Solution*. Unlike Treude et al. [6] and Nasehi et al. [8], in our work we did a categorization of SO's Q&A pairs instead of SO's questions.



DOI: 10.1002/smr

Our work also lies in the field of search engines and code sample retrieval from the Web. Umarji et al. [38] investigated developers' habits in searching code on the Internet. Sim et al. [39] investigated how sites for general purpose information retrieval (e.g., Google) outperform custom sites for code search (e.g., Krugle) and component reuse (e.g., SourceForge) in retrieving code samples from the Internet. In our approach we perform code retrieval on SO. When samples are retrieved from search engines, the developer has to assess their validity. Since we rely on SO, the code samples are already assessed by the community.

Holmes et al. discussed in their study that often developers become lost when trying to use an API, unsure of how to make progress on a programming task. They presented STRATHCONA [40], a plug-in for the Eclipse IDE that uses the structure of the source code under development to find relevant examples in a repository. Similarly, our approach addresses the same problem but using another strategy: harnessing the "crowd knowledge" available in SO to recommend information that can assist developers to solve programming tasks using a given API.

Ponzanelli et al. [34] 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 [41], an Eclipse plugin which assists developers in discovering code examples on the web related to their current task. FISHTAIL suggests code examples according to the most changed program entity name. In some of the activities (*Scenario KAE*) used to evaluate our approach, we also focused on entity names (e.g., class names) using them as part of the search used to query our system. In the presented approach, we perform code retrieval on SO, and because of that we could rely on the code samples being already assessed by the community.

HIPIKAT [42] is a recommendation system developed to support newcomers in a project by recommending items from problem reports, newsgroup, and articles. Our approach focus on recommending content from SO instead of providing resources from in-project knowledge.

Cordeiro et al. [43] 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 the index previously created from Q&A pairs.

Takuya et al. presented SELENE [44], 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, which already contain developer explanation that enriches code snippet examples.

Brandt et al. presented BLUEPRINT [45], a plugin built on top of Adobe Flex Builder that allows developers to search and import code examples in the IDE. Similarly, Zagalsky et al. presented EXAMPLE OVERFLOW [46] a web-based tool to search and recommend JavaScript code samples. In our work, we give the freedom of searching for code samples of any programming language.

## 8. CONCLUDING REMARKS AND FUTURE WORK

We presented a novel approach to leverage the Q&A crowd knowledge. This strategy recommends a ranked list of question-answer pairs from SO. The ranking criteria takes into account the textual similarity of the pairs with respect to the developer's problem, the quality of the pairs, and their *How-to* characterization. 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. We can state that $\text{NDCG}_{Relev}$ for Google was 5.62%



DOI: 10.1002/smr

higher than *Lucene+Score+How-to* approach and $\text{NDCG}_{Reprod}$ for Google was 4.75% higher than *Lucene+Score+How-to* approach. These results show that the overall quality of the recommended pairs by Google was slightly better than the *Lucene+Score+How-to* approach. Concerning our approach, 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 suggest that our approach outperforms the results obtained in a related work [34].

We also performed a comparison with Google built in search engine. The *Lucene+Score+How-to* approach achieved better performance than Google on Boost. Moreover, *Lucene+Score+How-to* obtained a number of poor results on Boost and LINQ topics (32.08% and 31.81%, respectively) lower than Google (46.66% and 40.90%, respectively). On the other hand, Google had performed better than *Lucene+Score+How-to* approach on Swing.

The Google search engine has two important features that help it produce high precision results. First, it makes use of the link structure of the Web to calculate a quality ranking for each web page. This ranking is called PageRank [47]. Second, Google makes use of the anchor text to improve search results [48]. This has several advantages. First, anchors often provide more accurate descriptions of web pages than the pages themselves. Second, anchors may exist for documents which cannot be indexed by a text-based search engine, such as images, programs, and databases. These two features are not present in Lucene's search engine. Moreover, Lucene is not efficient in processing documents with larger text fields because of the cost involving with the number of iterations through the input stream, which must be done to yield valid tokens [49].

Thus, we suggest that the ideal approach for recommendation is to use the Google built in search engine jointly with the proposed *How-to* classifier. We observed that some SO posts recommended by Google were more theoretical (i.e., conceptual posts) than practical (i.e., posts that show step by step how to solve a desired programming task with a given API). We suggest that the absence of a *How-to* classifier had contributed to this undesired scenario for Google.

Another problem with Google's recommendation called us attention because we defined that the search query must contain the API name. Despite this, Google returned results that were not related to the respective search API. Our proposed approach does not suffer from this problem because we filtered only SO posts related with the API of interest using the tag field. Thus, we suggest that this "hybrid approach" could use the Google search engine instead of Lucene and then use the *How-to* classifier to post-filtering the "How-to posts" by tag (i.e., the tag's name corresponds to the name of the search API).

As future work, other classifiers and other attributes could be investigated in order to improve the success rate and thus, the overall ranking of our recommendation approach. Moreover, new forms of query generation and respective indexing could be designed. For instance, the query could be extracted from the contextual source code where the developer is working. We could also investigate new search mechanisms beyond Lucene to improve the ranking.

Another future work can be performed with less popular topics to investigate the impact of the popularity of the topic in the result. For pretty popular topics, the chance of having good SO answers is higher. Even subjects that are in cookbooks may present variability in their popularity.

## 9. ACKNOWLEDGMENTS

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

## REFERENCES

[1] Parnin C, Treude C, Grammel L, Storey MA. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. *Georgia Tech, Tech. Rep.* 2012; (GIT-CS-12-05), doi:10.1.1.371.6263.

[2] Mockus A, Fielding RT, Herbsleb J. A case study of open source software development: the Apache Server. *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, ACM Press, 2000; 263–272, doi:10.1.1.178.4816.

[3] Mamykina L, Manoim B, Mittal M, Hripcsak G, Hartmann B. Design lessons from the fastest Q&A site in the west. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM: New York, NY, USA, 2011; 2857–2866, doi:10.1145/1978942.1979366. ISBN: 978-1-4503-0228-9.

[4] Ponzanelli L, Bacchelli A, Lanza M. Seahawk: Stack overflow in the IDE. *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, IEEE Press: Piscataway, NJ, USA, 2013; 1295–1298.

[5] Barzilay O, Treude C, Zagalsky A. *Facilitating Crowd Sourced Software Engineering via Stack Overflow*. Springer: New York, 2013; 297–316, doi:10.1007/978-1-4614-6596-6_15.

[6] Treude C, Barzilay O, Storey MA. How Do Programmers Ask and Answer Questions on the Web? (NIER track). *Proceedings of the 33rd International Conference on Software Engineering*, ACM, 2011; 804–807, doi:10.1145/1985793.1985907. ISBN: 978-1-4503-0445-0.

[7] de Souza LBL, Campos EC, Maia MA. Ranking Crowd Knowledge to Assist Software Development. *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC 2014, ACM: New York, NY, USA, 2014; 72–82, doi:10.1145/2597008.2597146.

[8] Nasehi S, Sillito J, Maurer F, Burns C. What makes a good code example? A study of programming Q&A in Stack Overflow. *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, 2012; 25–34, doi:10.1109/ICSM.2012.6405249.

[9] Pohar M, Blas M, Turk S. Comparison of logistic regression and linear discriminant analysis : A simulation study. *Metodoloki Zvezki* 2004; **1**(1):143–161.

[10] le Cessie S, van Houwelingen J. Ridge Estimators in Logistic Regression. *Applied Statistics* 1992; **41**(1):191–201, doi:10.2307/2347628.

[11] Linares-Vásquez M, Mcmillan C, Poshyvanyk D, Grechanik M. On Using Machine Learning to Automatically Classify Software Applications into Domain Categories. *Empirical Software Engineering* Jun 2014; **19**(3):582–618, doi:10.1007/s10664-012-9230-z.

[12] Haykin S. *Neural Networks: A Comprehensive Foundation*. 2nd edn., Prentice Hall PTR: Upper Saddle River, NJ, USA, 1998. ISBN: 0132733501.

[13] Sehgal L, Mohan N, Sandhu PS. Quality Prediction of Function Based Software Using Decision Tree Approach. *International Conference on Computer Engineering and Multimedia Technologies (ICCEMT)*, 2012; 43–47.

[14] Lempitsky V, Verhoek M, Noble A, Blake A. Random Forest Classification for Automatic Delineation of Myocardium in Real-Time 3D. *Functional Imaging and Modeling of the Heart*, Springer Berlin Heidelberg, 2009; 447–456, doi:10.1.1.150.1029.

[15] Dietterich T. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.* Sep 1995; **27**(3):326–327.

[16] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 2009; :10–18doi:10.1145/1656274.1656278. ACM.

[17] Karaa W, Griba N. Information retrieval with Porter stemmer: A new version for English. *Advances in Computational Science, Engineering and Information Technology*, *Advances in Intelligent Systems and Computing*, vol. 225. Springer International Publishing, 2013; 243–254.

[18] Porter MF. Readings in Information Retrieval. chap. An algorithm for suffix stripping, Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1997; 313–316.

[19] Yang Y, Pedersen JO. A Comparative Study on Feature Selection in Text Categorization. *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1997; 412–420. ISBN: 1-55860-486-3.

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

[21] Mcnemar Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 1947; **12**:153–157.

[22] Bostanci B, Bostanci E. An Evaluation of Classification Algorithms Using Mc Nemar's Test. *Proceedings of Seventh International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, vol. 201. Springer India, 2013; 15–26, doi:10.1007/978-81-322-1038-2_2.

[23] Liddell D. Practical tests of 2x2 contingency tables. *Journal of the Royal Statistical Society* 1976; **25**:295–304.

[24] Eckstein R, Loy M, Wood D. *Java Swing, year = 1998, note = ISBN: 1-56592-455-X, publisher = O'Reilly Media, address = Sebastopol, CA, USA*.

[25] Darwin IF. *Java Cookbook*. O'Reilly Media: Sebastopol, CA, USA, 2004. ISBN: 978-0-596-00701-0.

[26] Polukhin A. *Boost C++ Application Development Cookbook*. Packt Publ.: Birmingham, 2013. ISBN: 1849514887 9781849514880.

[27] Hilyard J, Teilhet S. *C# 3.0 Cookbook, 3rd Edition*. Third edn., O'Reilly, 2007. ISBN: 9780596516109.

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

[29] Holmes R, Cottrell R, Walker RJ, Denzinger J. The end-to-end use of source code examples: An exploratory study. *Proceedings of the IEEE International Conference on Software Maintenance*, 2009.

[30] Robillard MP. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 2009; **26**(6):27–34.

[31] Subramanian S, Holmes R. Making Sense of Online Code Snippets. *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, IEEE Press: Piscataway, NJ, USA, 2013; 85–88.

[32] Cohen J. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. 1968, doi:10.1037/h0026256.

[33] Manning CD, Raghavan P, Schütze H. *Introduction to Information Retrieval*. Cambridge University Press: New York, NY, USA, 2008. ISBN: 0521865719 9780521865715.

[34] Ponzanelli L, Bacchelli A, Lanza M. Leveraging Crowd Knowledge for Software Comprehension and Development. *CSMR*, Cleve A, Ricca F, Cerioli M (eds.), IEEE Computer Society, 2013; 57–66, doi:10.1109/CSMR.2013.16. ISBN: 978-1-4673-5833-0.

[35] Wang S, Lo D, Vasilescu B, Serebrenik A. EnTagRec: An enhanced tag recommendation system for software information sites. *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2014.

[36] Golder SA, Huberman BA. Usage patterns of collaborative tagging systems. *J. Inf. Sci.* Apr 2006; **32**(2):198–208.

[37] Xia X, Lo D, Wang X, Zhou B. Tag recommendation in software information sites. *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, IEEE Press: Piscataway, NJ, USA, 2013; 287–296.

[38] Umarji M, Sim SE, Lopes CV. Archetypal internet-scale source code searching. Milan, Italy, 2008; 7.

[39] Sim SE, Umarji M, Ratanotayanon S, Lopes CV. How well do search engines support code retrieval on the web? *ACM Trans. Softw. Eng. Methodol.* Dec 2011; **21**(1):4:1–4:25, doi: 10.1145/2063239.2063243.

[40] Holmes R, Walker RJ, Murphy GC. Approximate Structural Context Matching: An Approach to Recommend Relevant Examples. *IEEE Trans. Softw. Eng.* Dec 2006; **32**(12):952–970, doi: 10.1109/TSE.2006.117.

[41] Sawadsky N, Murphy GC. Fishtail: From Task Context to Source Code Examples. *Proceedings of the 1st Workshop on Developing Tools As Plug-ins*, ACM: New York, NY, USA, 2011; 48–51, doi:10.1145/1984708.1984722. ISBN: 978-1-4503-0599-0.

[42] ČubraniĆ D, Murphy GC, Singer J, Booth KS. Learning from Project History: A Case Study for Software Development. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, ACM: New York, NY, USA, 2004; 82–91, doi: 10.1145/1031607.1031622. ISBN: 1-58113-810-5.

[43] Cordeiro J, Antunes B, Gomes P. Context-based Recommendation to Support Problem Solving in Software Development. *Proceedings of 3rd Int. Workshop on RSSE)*, IEEE, 2012; 85–89, doi:10.1109/RSSE.2012.6233418.

[44] Takuya W, Masuhara H. A Spontaneous Code Recommendation Tool Based on Associative Search. *Proceedings of the 3rd International Workshop on Search-Driven Development*, ACM, 2011; 17–20, doi:10.1145/1985429.1985434. ISBN: 978-1-4503-0597-6.

[45] Brandt J, Dontcheva M, Weskamp M, Klemmer SR. Example-centric programming: Integrating web search into the development environment. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, ACM: New York, NY, USA, 2010; 513–522, doi:10.1145/1753326.1753402.

[46] Zagalsky A, Barzilay O, Yehudai A. Example Overflow: Using social media for code recommendation. *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*, IEEE, 2012; 38–42, doi:10.1109/RSSE.2012.6233407.

[47] Page L, Brin S, Motwani R, Winograd T. The PageRank Citation Ranking: Bringing Order to the Web. *Technical Report 1999-66*, Stanford InfoLab November 1999. Previous number = SIDL-WP-1999-0120.

[48] Brin S, Page L. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, The Netherlands, 1998; 107–117.

[49] Su DCC. Performance analysis and optimization on Lucene. *Technical report*, Stanford InfoLab 2002.



DOI: 10.1002/smr