# On the Extraction of Cookbooks for APIs from the Crowd Knowledge

Lucas B. L. de Souza
Department of Computer Science
Federal University of Uberlândia
Uberlândia, MG, Brazil
Email: lucas.facom.ufu@gmail.com

Eduardo C. Campos
Department of Computer Science
Federal University of Uberlândia
Uberlândia, MG, Brazil
Email: eduardocunha11@gmail.com

Marcelo de A. Maia
Department of Computer Science
Federal University of Uberlândia
Uberlândia, MG, Brazil
Email: marcmaia@facom.ufu.br

***Abstract*—Developers of reusable software elements, such as libraries, usually have the responsibility to provide comprehensive and high quality documentation to enable effective software reuse. The effective reuse of libraries depends upon the quality of the API (Application Program Interface) documentation. Well established libraries typically have comprehensive API documentation, for example in Javadocs. However, they typically lack examples and explanations, which makes the effective reuse of the library difficult.**

**StackOverflow.com (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 solving their problems. Despite the increasing adoption of SO, the information related to a particular topic is spread along the website. Thus, SO still lacks an organization of its crowd knowledge.**

**In this paper, we present an automatic approach that organizes the information available on SO in order to build cookbooks (recipe-oriented books) for APIs. In the experiments conducted to test our approach, we generated cookbooks for three APIs widely used by the software development community: SWT, STL and LINQ. We have defined some criteria to test the cookbooks' quality. The results have shown that for the studied APIs, the cookbook proved to have a meaningful chapter organization, since 63–71% of the chapters have a defined meaning. Moreover, 72–88% of the recipes are related to the meaning of its containing chapter. In addition, we also verified the quality of the individual question and answers pairs presented in the cookbooks, regarding three properties: the appropriateness to be part of a cookbook, the self-containment and the reproducibility of its source code snippets. We have shown that 65–75% of the pairs met, at least partially, these properties.**

## I. Introduction

Accordingly to Parnas (qtd. in Brooks [1] p.224) "Reuse is something that is far easier to say than to do. Doing it requires both good design and very good documentation. Even when we see good design, which is still infrequently, we won't see the components reused without good documentation.".

Application Programming Interfaces (APIs) are exposed to developers in order to allow the reuse of software libraries [2]. Traditionally, many kind of software documentation, like API documentation, are generated by few people and have a target audience much larger. The resulting documentation, when exists, generally has poor quality and lacks examples and explanations [3]. Although developers of reusable software elements have the responsibility to provide high quality and comprehensive documentation [4][2], we can observe that the official documentation is not the unique source of information that developers handle during the reuse process. Because developers often do not have all the information they need to complete a particular task using a library, their closer alternative is to search the Web. They look for information that will help them solve their software development problems [5]. In the last decade, concomitantly with the emerging of Web 2.0, a new culture and philosophy emerged and are 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) [3]. Similar to the way that open source development has changed the traditional process of software development [6], these new forms of collaboration and contribution have the potential to redefine how developers learn, preserve and share knowledge about software development. This phenomena may also corroborate to the fact that developers of reusable software elements still fail to attach the proper importance to documentation.

Currently, one of the main source of information for that kind of search is StackOverflow – SO, a Question and Answer (Q&A) website related to software development issues. SO uses social media to facilitate knowledge exchange between programmers by attenuating the pitfalls involved in using source code from the web [5]. Accordingly to Barzilay et al. [5], "Its design nurtures a community of developers, and enables crowd sourced software engineering activities ranging from documentation to providing useful, high quality code snippets". The set of information available on this social media services is called "crowd knowledge" and often become a substitute for the official software documentation [7].

Nonetheless, the use of the "crowd knowledge" for a substitute or a complement for the documentation has its own drawbacks. Considering SO, we can observe that the information related to a particular API is spread along the website. For instance, there are more than 2500 threads on SO related to the API SWT. Although it is possible to search those threads to get relevant information for a specific task using either the internal SO query mechanism or even using general purpose search engines, there is no available notion of structured content that the developer could follow in a meaningful way, such as, a book or a tutorial.

In this work, our goal is to propose an approach to organize part of the "crowd knowledge" by creating a kind of structured documentation known as *cookbook*. A cookbook is composed of chapters on a specific theme and each chapter is filled with a set of recipes. Each recipe contains a problem that can be solved with elements of that API and also contains instructions on how to solve that problem.

The cookbooks generated by our approach are meant to be used in a different way than Q&A sites. If a developer has a specific task at hand that needs to be solved, a query that describes the problem must be formulated and given to the Q&A site, which recommends threads that can be useful to help the solution of the problem. On the other hand, cookbooks can be used by someone who wants to know what are the main subjects of an API. This can be specially useful to someone that is new to that API and want to have a broad vision of it. This person could read the titles of chapters to check what are the main topics concerning the API and then could focus on those topics considered more interesting. In other words, cookbooks have a exploratory characteristic, instead of search-driven one. The generated cookbooks could also even be used as a starting point to build edited versions of a cookbook for some API, since in our approach we aim at identifying the main concepts of an API and identifying high quality content to fill the chapters.

In order to evaluate the effectiveness of our approach, we generated cookbooks for three popular APIs widely used by the software development community, related to different programming languages: SWT[1] (Java), STL[2] (C++) and LINQ [8] (.NET languages). Some criteria to assess the generated cookbooks were defined. Firstly, we evaluated the quality of the chapters, analyzing in which extension its chapters are well defined, since ideally one chapter should be related to one specific theme of the API. We also checked if the recipes are related with the chapters where they were included. Finally, each question and answers pair presented in the cookbooks was evaluated considering the following three criteria: the *appropriateness* to be part of a cookbook, the *self-containment*, and the *reproducibility* of its code snippets. The raw cookbooks studied in this work can be seen at our website[3].

The rest of this paper is organized as follows. In Section II we present the process of construction of cookbooks. Section III presents the criteria used to evaluate the cookbooks. In section IV we present and discuss the results and threats to validity. The related work is discussed in Section V and the conclusions are presented in Section VI.

## II. Cookbook Construction

The first step necessary for the development of this work was to obtain the data dump that the SO team provides on its blog[4] . The data dump contains the entire contents available on SO since its creation (in 2008) until the building date of the dump, including posts (questions and answers), users, votes, etc. We have used the version released in March 2013. This dump was imported into a relational database that was used in the construction of cookbooks.

[1]http://www.eclipse.org/swt/

[2]http://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm

[3]http://lascam.facom.ufu.br

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

### A. Types of Questions in SO

In SO, users ask many kind of different questions. Accordingly to Nasehi et al. [9] "SO question types can be described based on two different dimensions. The first dimension deals with the question's 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 build a cookbook for SWT library, we only consider in our approach threads in which the question has the tag "SWT". A thread of discussion in SO is formed by a question and a series of answers to that question. In the line "#Questions" of Table I, we show the number of questions (that is equal to the number of threads because each thread has one question) for the three APIs considered in this paper. Yet accordingly to Nasehi et al. "questions from SO can also be classified in a second dimension that is about the main concerns of the questioners and what they wanted to solve". In this dimension, Nasehi et al. identified four categories: *Debug-Corrective*, *Need-To-Know*, *How-To-Do-It* and *Seeking-Different-Solution*. They defined the *How-To-Do-It* category as the one in which the questioner provides a scenario and asks about how to implement it, which is very close to the idea of recipes from cookbooks, since they also have the goal to provide instructions on how to perform a task concerning the respective API. Thus, in our approach, we only consider threads whose questions are in the category *How-To-Do-It*. In order to identify this type of questions, we developed a simple, yet automated, rule-based approach that considers the terms present in the question's title and body. A question is classified as a *How-To-Do-It* if the three following conditions are satisfied:

- It has the term "how" in its title or body;
- It does not have in its body the presence of terms generally related to the *Debug-Corrective* category: "fail", "problem", "error", "wrong", "fix", "bug", "issue", "solve", "trouble";
- It does not have the word "error" in its code snippets (if any are present).

TABLE I: SO's Data Information.

| Metric | SWT | STL | LINQ |
|---|---|---|---|
| #Questions | 2243 | 5562 | 26869 |
| #How-To's | 752 | 1188 | 8787 |
| #How-To Pairs (NP(A)) | 1089 | 3738 | 20083 |

Using these rules, we tried to differentiate between the categories *How-To-Do-It* and *Debug-Corrective*, since for this last category the word "how" generally is also used, but with the intention to ask help on how to fix a problem in the question's code snippet. The words used in the rules described above were defined after analyzing a random sample of 70 questions from SO. In the evaluation of the cookbooks created using our approach, one of the defined criteria (*Appropriateness*) is used to check if the questions selected to be part of a cookbook are indeed *How-To-Do-It*, which allowed us to test the performance of this simple approach. The line "#How-To's" in Table I shows the number of questions that were classified as *How-To-Do-It*. For instance, 752 of the 2243 questions that have the tag "SWT" were considered *How-To-Do-It*. As the number of answers is different among different threads, each thread originates a different number of Q&A pairs (if a thread has *n* answers, there are *n* possible pairs for that thread, and each pair is composed by an answer and the question of the thread). In the line "#How-To Pairs (NP(A))", we showed the number of Q&A pairs that can be retrieved using the questions classified as *How-To-Do-It*. Suppose that *NP* is a function that returns these number of pairs for an API A. For instance, *NP(SWT)* = 1089.

## *B. Ranking of Q&A Pairs*

In the generation process of cookbooks, our goal is to select only good quality content, i.e., posts well evaluated by the SO crowd. Here we consider that the score of a post (question or answer) is a proxy for its quality because the voting mechanism on SO is the main feature that allows SO members evaluate its content. A score of a post is the difference between the upvotes and downvotes it received.

In order to rank the content about an API in our database and then, to select only the best part of it to fill the cookbook, a ranking of Q&A pairs was produced. We decided to rank Q&A pairs instead of entire threads because the answers for the same question can have different scores, i.e., some answers can be better than others. Thus, the ranking of pairs allows us to differentiate well voted from poor voted answers for a same question. Each Q&A pair in this ranking is composed by a question and one answer for that question.

Because each individual post on SO has its own score and we want to rank Q&A pairs, 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. We decided not to follow this approach because the answer seems to be more important than its respective question. The reason for our hypothesis is that the answer usually carries more information about the problem's solution. Thus, we decided to consider the score of pair as the weighted mean value between the individual scores of its answer and question. We arbitrarily defined the values 7 and 3 for the weights of the answer and question from a pair. We agree that different choices would lead to different results, but we expected to achieve acceptable results with this choice. Given an API (e.g., SWT), we can calculate the SO score of each *How-To-Do-It* pair that belong to that topic. Then, we can rank the set of pairs in descending order.

Because we want to select the pairs that are on the top of the ranking, the definition of a threshold is necessary. With this goal, we defined the equation shown in (1), where *MP(A)* represents the maximum position allowed for an API *A* (i.e., the threshold that we are looking for); *NP(A)* represents the number of *How-To-Do-It* pairs that belong to API *A* (information shown in Table I); *min* is a function that returns the smaller of two values; *(int)* is just a cast to integer type. The interpretation for this formula is to obtain the threshold by taking the smaller value between 400 and 10% the number of pairs of the API. The reasoning behind this equation is that for APIs with large number of #How-To pairs (e.g., LINQ that has 20083 pairs) the threshold calculated will be 400, since 10% of the pairs of the API (in the case of LINQ, is 2008) is too much information to be included in a cookbook. This strategy limits the size of the final cookbook, since it could make its reading and exploration difficult. Moreover, *comprehensiveness* is not a goal for our cookbooks. This decision is based on the fact the the crowd may not produce comprehensive content, so it does not make sense to evaluate our approach from this point of view. Our goal is mainly to produce a cookbook with high quality content, which size is similar as the human-edited cookbooks. For APIs that are smaller (e.g., SWT that has 1089 pairs), the threshold is 10% of the number of its pairs (108 in case of SWT). Table II shows the thresholds calculated for the three APIs considered in the experiments. These thresholds are used in the algorithm for building cookbooks.

$$MP(A) = (int)min(400, 0.1 * NP(A)) \tag{1}$$

TABLE II: Maximum Position Allowed per API.

| API (A) | Maximum Position (MP(A)) |
|---|---|
| SWT | 108 |
| STL | 373 |
| LINQ | 400 |

## *C. Identification of Chapters*

We used the topic modeling technique Latent Dirichlet Allocation (LDA) [10] to identify the chapters of a cookbook, because its the widely used technique to find discussion topics in natural language text documents. Each topic found by LDA potentially originates a chapter in the cookbook being generated.

Before applying LDA, the data is pre-processed similarly to the work conducted by Barua et al. [11]. For each thread, where the question is classified as *How-To-Do-It*, we create a document containing the textual content of the question and all its answers (i.e., the content of question's title, question's body and answers's bodies). We preprocess the content of these documents before applying LDA. Firstly, we discard the code snippets that are present in the posts (if any), because source code syntax (e.g., "while" and "for" loops) introduces noise into the analysis phase, since all code snippets contain similar programming language syntax and keywords, that do not help topic models to find meaningful topics [11], [12]. We also remove all HTML tags (e.g., <br>and <a href="..."), since our focus is to analyze natural language (English) content. Later, we remove common English words (stop words). Finally, we apply the Porter stemming algorithm [13], to map the words to their base form (e.g., "programming", and "programmer" both get mapped to "program"). We used Apache Lucene[5] to remove stop words and stemming. Furthermore, we used the Java library HTML Cleaner[6] to remove HTML tags.

Each document generated after this pre-processing was included in the corpus of documents used to run LDA. As for each thread we generated one document, the size of the

[5]http://lucene.apache.org/core

[6]http://htmlcleaner.sourceforge.net

corpus coincides with the values shown in the line "#How-To's" of Table I. For instance, the size of the corpus for API SWT is 752.

The number of topics (*K*), is a user-specified parameter for LDA that provides control over the granularity of the discovered topics. Smaller values of *K* generates more general topics and larger values of *K* generates more detailed topics. There is no unique value of *K* that is suitable in all situations and all datasets [14]. In order to determine the number of topics, i.e., the value of parameter *K* for an API *A* (in short, *K(A)*), we defined the equation shown in (2). This formula defines *K(A)* as the minimum value between 20 and 15% of the threshold calculated by equation (1) (i.e., *MP(A)*). The reasoning behind this formula is that APIs with a large amount of pairs (e.g., LINQ with 20083) will have *K(A)* = 20, i.e., our intended upper bound for the number of chapters in a cookbook. In contrast, for smaller APIs such as SWT (1089 pairs), the *K(A)* will be calculated as a percent of the position threshold calculated before (for SWT, *K(A)* = 16). This is a way to limit the number of topics to 20 in the case of the most popular APIs, which we believe is a reasonable number of topics in a cookbook for this kind of APIs. In contrast, we believe that 20 topics would be too much for APIs with less content on SO (e.g., SWT), so we calculate the number of topics as a percent of the threshold calculated by formula (1). Table III shows the number of topics calculated for each API considered in the experiments.

$$K(A) = (int)min(20, 0.15 * MP(A)) \quad (2)$$

TABLE III: Number of Topics per API.

| API (A) | Number of Topics (K(A)) |
|---|---|
| SWT | 16 |
| STL | 20 |
| LINQ | 20 |

After determining the *K* value, we run LDA for 2000 Gibbs sampling iterations, which is sufficient for the Gibbs sampling algorithm to stabilize [15]. We used the LDA implementation available in library MALLET[7]

After running LDA, the following information is available:

- There are *K* topics, and for each one exists some top terms that describes it. For instance, after running LDA for SWT, one of the topics had among its top terms "tabl, column, row, cell". Observe that this terms are stems instead of entire words ("tabl" instead of "table"), because one of the steps before applying LDA was stemming the textual content of the threads. This set of top terms is used as the title of the chapters. Currently for each chapter's title we show 20-top terms of its corresponding topic. The number 20 has showed in our experiments to be sufficient for the identification of the chapter's meaning;
- For each document (*D*) that composes the corpus used in the LDA (remember that each document corresponds to a SO thread), we have its dominant topic, i.e., the topic that is most related to it. Consider that *getDominantTopic(D)* is a function that receives a document and returns its dominant topic.

## *D. Cookbook Generation*

Now we present the algorithm used to build a cookbook *C* for an API *A*. The pseudocode is shown in Algorithm 1. For each document *D* present in the corpus used in LDA, we retrieved its dominant topic. Then we retrieved all pairs that can be built from that thread. For each pair, we retrieved its position in the ranking of pairs for API *A* and then we checked several conditions and if all of them are satisfied then the pair is eligible to be included in the cookbook into the chapter corresponding to the dominant topic of its thread. The verified conditions are:

- The position of the pair in the ranking by score, must not be superior to the maximum rank position allowed (*MP(A)*). This is a way of ensuring that only well evaluated pairs were included in the cookbook;
- The answer must have source code snippets. We identify the presence of snippets trough the use of the HTML tags "<pre><code>...". We require the presence of source code in the answers, because programming by examples is a intuitive way to learn both for novices and experts [16];
- The answer and question that composes the pair must not have dead links in its content. This a way to ensure the external sources referenced in the posts can still be accessed. Table IV shows for the three APIs, the number of different links and the number of dead links present in the pairs belonging to it. In order to verify if a link is dead or not, we used the Java library HttpUnit[8] ;
- The question must not be too large. Questions with much content (verbose questions) usually contains many queries (i.e., the questioner asks many things) or have a difficult to understand problem. We decided to not include this type of questions, to make cookbooks easier to read and understand. We defined a threshold for the maximum allowed size of a question, that considers the number of characters present in its body. Figure 1 shows a histogram with the percent of question's size in many sizes's ranges. The data considered in this graph are all questions for SWT, STL and LINQ. The questions with size less to 1300 characters comprises 81.4% of the question. Thus, we decided to not include pairs in the cookbook in which its question has size greater than or equal to 1300 characters.

Considering a thread (document), all their pairs that satisfy all conditions are added to a list. At the end, if the list contains only one pair, this pair is added to cookbook. If the list has more than one pair, the two pairs with larger score (best position in the ranking by score) are added to the cookbook inside the chapter corresponding to the dominant topic of the

[7]http://mallet.cs.umass.edu/

[8]http://httpunit.sourceforge.net/

**Algorithm 1:** Generation of Cookbook $C$ for API $A$

```
$C \leftarrow$ new Cookbook();
foreach $D \in Corpus$ do
    $dominantTopic \leftarrow getDominantTopic(D)$;
    $pairs \leftarrow getPairs(D)$;
    $listOfCandidatePairs \leftarrow newList()$;
    foreach $P \in pairs$ do
        $rankingPosition \leftarrow getRankingPosition(P)$;

        if $rankingPosition \leqslant MP(A)$ then
            $Q \leftarrow getQuestion(P)$;
            $A \leftarrow getAnswer(P)$;
            if $hasSourceCode(A) \wedge$
              $doesNotHaveDeadLink(Q) \wedge$
              $doesNotHaveDeadLink(A) \wedge$
              $size(Q) < 1300$ then
                $listOfCandidatePairs.add(P)$;
$includePairsInCookbook(dominantTopic,$
                                  $listOfCandidatePairs, C)$
```

TABLE IV: Number of links per API.

| API (A) | Number of Links | Number of Dead Links |
|---|---|---|
| SWT | 2471 | 150 |
| STL | 5430 | 210 |
| LINQ | 13756 | 708 |

pair's thread. The final number of chapters in the cookbook can be smaller than the number of topics ($K$) used as input for LDA run because it is possible that for a topic found by LDA, none of the pairs have satisfied the conditions of the cookbook generator algorithm. The empty topics do not originate chapters in the cookbook. The first line of Table V shows the number of chapters for the three APIs. For instance, the number of topics calculated in LDA for SWT was 16, but only 14 chapters had some content.

The pairs included into the chapters of a cookbook are organized into *Recipes*. Each recipe is a set of pairs in which the pair's question coincides. As we only add at most two pairs from the same thread, the size of this set is one or two. In others words, a recipe is a question, with one or two solutions. The reason to allow more than one solution per question, it that the solutions contained in the answers can be different, so it is interesting to provide more than one alternative to the user. The explanation to provide at most two solutions, is to avoid too large recipes. Moreover, the solutions showed (one or two solutions) were well evaluated by the SO crowd. Thus, using the solutions provided in at most two answers must be enough to solve the problem described in the question. The second and third lines of Table V show, respectively, the number of pairs and recipes in the three generated cookbooks.

## III. Evaluation Criteria

In this section, we present the criteria used to evaluate the cookbooks generated by our approach. All criteria range from 0 (worst value) to 4 (best value).

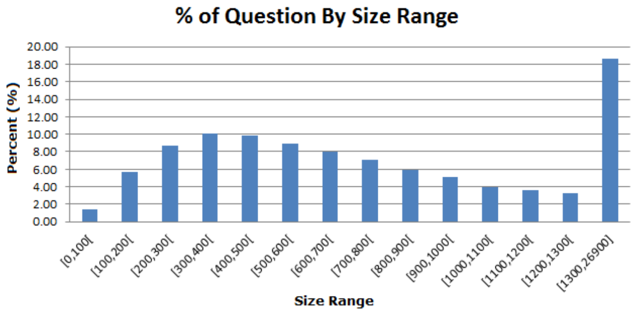

Fig. 1: Percent of Questions by Size Range.

TABLE V: Cookbooks Metrics.

| Metric | SWT | STL | LINQ |
|---|---|---|---|
| Number of Chapters | 14 | 19 | 20 |
| Number of Pairs | 53 | 189 | 202 |
| Number of Recipes | 48 | 152 | 152 |

Firstly, we assess the chapter organization of the cookbooks using two criteria. The first criterion (*Semantics of Chapter* or *SmtChap*) was used to verify to what extent the meaning of the titles of the chapters were well defined. The value 4 means a perfectly defined subject in the title and value 0 means that no subject or meaning could be found by reading the title. The meanings found were also annotated. This criterion is not boolean because for some chapters is it possible to identify a subject, however it's mixed with other minor subjects and/or unrelated terms. After rating *SmtChap* for all chapters, the ones with *SmtChap* greater than or equal to 3, have their recipes analyzed to check to what extent each recipe is related to main subject found for its chapter during evaluation of *SmtChap*. This criterion is called *Conformity to Chapter* or *ConfChap*. The reason behind this second criterion can be explained using an example. One of the chapters of LINQ cookbook had a very well defined meaning (*SmtChap* equal to 4) that was annotated as "sorting of lists". However, one of the recipes inside it was not related to sorting. In that recipe the word "order" was used many times, but in the context of a "sale order". In this case, the value for *ConfChap* was 0 because although the word "order" is related to sorting, in that recipe it was being used with other meaning.

The second part of the evaluation has a smaller granularity: we assess three criteria for each pair (one recipe can have one or two pairs). The first one is the *Appropriateness* (in short, *Approp*) to be part of a cookbook for its API. This criteria is a way to test the rule-based approach explained in Subsection II-A, because we check if the pair's questions are *How-To-Do-It*, since questions belonging to other categories are not desirable in cookbooks, just like *Debug-Corrective* (that contains buggy code) or more theoretical questions – the cookbooks are aimed to contain practical activities. This criterion is also used to evaluate if the solution presented in the answer of the pair is using the target API of the cookbook (e.g., for some recipes in cookbook for STL, although the questioner explicitly asked for a solution using STL, the community only gave solutions using BOOST libraries). The next analyzed criterion was *Self-containment* (*SelfCont*). In this criterion, we checked to what extent the information referenced in the pair is self-contained. Although many posts have links to external sources (e.g., blogs, tutorials, official API documentation), we want to avoid the cases in which the solution is just presented via an URL link, because the links can become unavailable someday. Thus, it is important to replicate the information that is important for the problem's solution inside SO, even if the solution is already presented in a external source. The next criterion was called *Reproducibility* (*Reprod*). This criterion was used to evaluate to what extent the source code snippets available on the question and answer bodies of a pair can be easily compiled and executed. The grade also ranges from 0 to 4. The value 0 means that 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).

We present an evaluation composed of experiments with the cookbooks generated for the three considered APIs to assess their quality. We made a qualitative manual analysis of the chapters and recipes of the cookbooks, in order to rate the considered criteria. The evaluations were done by the first two authors of this paper with a double-check mechanism.

## IV. Results and Discussion

In this section we present and discuss the results regarding the evaluation of the cookbooks for the three considered APIs. In Subsection IV-A we discuss aspects related to the organization of chapters in cookbooks, considering the criteria *SmtChap - Semantics of Chapter* and *ConfChap - Conformity to Chapter* and also the chapters' sizes. Subsection IV-B deals with the evaluation of the individual Q&A pairs included in the cookbooks. In Subsection IV-C, we present the threats to the results' validity.

### A. *Chapter's Evaluation*

The graphs in Figure 2 show for the three considered APIs, the grades given by the evaluators for criterion *SmtChap*. For SWT, four chapters have grade 2, four chapters have grade 3 and six chapters have grade 4. For STL, three chapters have grade 0, two have grade 1, two have grade 2, three chapters were graded as 3 and nine chapters received grade 4. For LINQ, one chapter have grade 0, one chapter was graded as 1, five chapters were graded as 2, eight received grade 3 and five have grade 4. We can observe that not every chapter has a corresponding grade because the identifier of each chapter is the one generated by LDA for its respective topic. Some topics do not have corresponding chapters in the cookbooks, because the chapters with no pairs are removed from the cookbooks. The results show that 71.43% (10 of 14), 63.16% (12 of 19) and 65% (13 of 20) of the chapters for SWT, STL and LINQ, respectively, were graded with *SmtChap* $\geq 3$, which are reasonable results, as more than 60% of all chapters proved to have a defined meaning.

Table VI shows the manually assigned meaning of the chapters, with *SmtChap* equals to 4, considering the cookbook for STL. Because of space constraints, we are not showing this information for the other two cookbooks. As can be seen from the table, the meanings assigned to the chapters are varied, covering different areas of STL library.

Table VII shows for the generated cookbooks, the distribution of chapters' size (number of recipes) considering a 5-range. In the table, we can see, for instance, that there are 13 chapters in SWT's cookbook containing between one and five recipes. The major part of the chapters has at most five recipes, indicating that our approach tends to produce small chapters. Nonetheless, there are some big chapters. One possible explanation for this variation is that for an API some subjects are more popular than others. Thus, the amount of information in SO may vary depending on the subject. For instance, chapter 17 of cookbook for STL has 18 recipes, and is about operations with arrays and vectors, which seems to be a very recurrent subject in STL. In the same cookbook, there is a chapter with *SmtChap* value equals to 0 (i.e., a chapter for which was not possible to identify a meaning) with 25 recipes. Thus, we decided to calculate the one-tailed Spearman's rank correlation coefficient (*rho*) between *SmtChap* value of each chapter and its number of recipes to check if chapters with low *SmtChap* value tend to have many recipes (i.e., to check if there is a strong negative relationship between the two variables). The results are shown in Table VIII. As the *p-values* found are greater than 0.05, thus, we cannot affirm that exists a negative relationship between the two variables, even though *rho* is negative for SWT and STL. Indeed, there are some chapters that although having more than 20 recipes, have *SmtChap* equals to four, indicating that our approach also has found big chapters that have well defined meaning.

The results concerning the evaluation of *ConfChap* are in Table IX, which shows the number of recipes that were graded with each 0-4 *ConfChap* values. As the goal of this metric is to evaluate the location of the recipe, these results are only about the chapters grade with *SmtChap* $\geq 3$, because only for a chapter having a defined meaning it is meaningful to check to which extent the recipes are related to it. The results show that for meaningful chapters, generally the recipes are related to the identified meaning of its chapter: 79.41% (27 of 34), 88% (88 of 100) and 72.38% (76 of 105) of the recipes belonging to chapters with *SmtChap* $\geq 3$, have *ConfChap* $\geq$ 3, for SWT, STL and LINQ, respectively. Overall, more than 70% of those recipes are related to its chapters. As can be seen from the table, almost all recipes have *ConfChap* values either 0 or 4 (i.e., they are not related at all or they are completely related to its chapter), which could suggest that *ConfChap* criterion should be boolean. Nonetheless, there are few recipes that have intermediary *ConfChap* values. These recipes are partially related to its chapters (e.g., one chapter for LINQ has the meaning 'Hierarchy in Trees' and one of its recipes is partially related to it, because although it used a method to get the children of a tree's node, the focus of the question was the operator "SelectMany"). The LINQ's cookbook has a higher percentage of recipes with *ConfChap* equals to zero (17.14%) than the other two cookbooks (14.71% and 5% for SWT and LINQ respectively). One fact that contributed to that is a chapter in LINQ's cookbook, for which was annotated the meaning "Iteration Over Lists" that had ten recipes not related

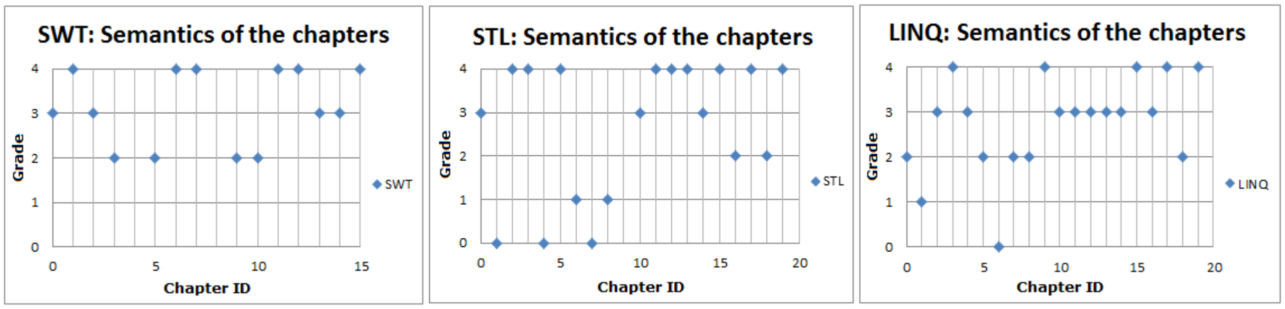

Fig. 2: Evaluation of *Semantics of Chapter* criterion.

TABLE VI: Meaning of STL Chapters with grade = 4.

| Chapter ID | Meaning |
|---|---|
| Chapter 2 | Aspects relating to compilation/pre-compilation and debug. |
| Chapter 3 | Aspects related to memory: allocation, release, heap, buffer, pool. |
| Chapter 5 | Operations with map data structure (e.g. iterate over a map). |
| Chapter 11 | Reading and Writing file using a buffer. |
| Chapter 12 | Working with constructors and destructors methods. |
| Chapter 13 | Operations on strings (e.g. length, substring). |
| Chapter 15 | Data Structures: queue, priority queue, red black tree, heap. |
| Chapter 17 | Operations with arrays and vectors (e.g. allocation, size). |
| Chapter 19 | Concurrence (Parallelism) using Thread Safe and Lock. |

to iteration. Those recipes were using lists in their solutions, however, were not iterating over then. In other case for LINQ, a recipe that uses the term "order" in the sense of "client order" was included in a chapter about "sorting of lists". As we rely on results from LDA, that is a syntactic (not semantic) technique, to include the recipes in the chapter, cases like the above can occur, however these cases are not common.

TABLE VII: Chapters' Size per API.

| Number of Recipes Range | SWT | STL | LINQ |
|---|---|---|---|
| [1,5] | 13 | 8 | 11 |
| [6,10] | 1 | 6 | 5 |
| [11,15] | 0 | 3 | 1 |
| [16,20] | 0 | 1 | 0 |
| [21,25] | 0 | 1 | 3 |

TABLE VIII: Spearman Rank Correlation Test Between *SmtChap* and Qty of Recipes.

| API | rho-value | p-value |
|---|---|---|
| SWT | -0.309 | 0.141 |
| STL | -0.146 | 0.276 |
| LINQ | 0.173 | 0.232 |

## *B. Q&A Pair's Evaluation*

The results concerning the evaluation of the three last criteria (*Approp*, *SelfCont* and *Reprod*) are shown in Table X. In this table, we defined a series of metrics to evaluate the quality of the pairs included in the cookbooks (remember that a recipe can have one or two Q&A pairs). Firstly, we measure the percent of pairs in a cookbook having a criterion (*Approp*, *SelfCont* or *Reprod*) greater or equal than 3 and then equal to 4. Then, we measured the percent of pairs having all the three criteria greater or equal than 3 and then equal to 4. As can be seen from the table, *Approp* criterion generally has reasonable values (mostly above 70%). The pairs with *Approp* lesser than 3 generally have one of the following features:

TABLE IX: Number of Recipes per *ConfChap* value.

| *ConfChap* Value | SWT | STL | LINQ |
|---|---|---|---|
| 0 | 5 | 5 | 18 |
| 1 | 0 | 3 | 3 |
| 2 | 2 | 4 | 8 |
| 3 | 0 | 4 | 1 |
| 4 | 27 | 84 | 75 |

- A question not belonging to the *How-To-Do-It* category. For instance, conceptual questions, where the questioner tries to understand some behavior of the API, and *Debug-Corrective* questions, in which the questioner needs help to fix a faulty source code. These cases represent false-positive questions of the rule-based approach developed to detect *How-To-Do-It* questions.
- An answer that do not contain a solution that uses the API aimed in the cookbook. For instance, there were some pairs in the cookbook for STL that presented solutions using BOOST library instead of STL library. Since it is a STL cookbook, is interesting that all pairs

present a solution that uses STL, even if they present other solutions using a different API. This is typically an inadequately tagged post.

TABLE X: Percent of Pairs per Metric.

| Metric | SWT | STL | LINQ |
|---|---|---|---|
| *Approp* $\geq 3$ | 90.56% | 70.90% | 82.17% |
| *Approp* = 4 | 86.79% | 69.31% | 81.19% |
| *SelfCont* $\geq 3$ | 100% | 95.77% | 97.52% |
| *SelfCont* = 4 | 100% | 94.18% | 97.52% |
| *Reprod* $\geq 3$ | 83.02% | 94.18% | 86.63% |
| *Reprod* = 4 | 56.60% | 70.37% | 47.52% |
| *Approp* & *SelfCont* & *Reprod* $\geq 3$ | 75.47% | 65.55% | 73.76% |
| *Approp* & *SelfCont* & *Reprod* = 4 | 45.28% | 45.50% | 40.09% |

The *SelfCont* criterion had very high values overall, meaning that almost all pairs included in the cookbooks were completely self-contained. In just a few cases, some answers posted a solution via a HTTP link, without giving any information about the solution in the answer itself. One hypothesis to explain this result is because we select only well evaluated pairs (high-scored) to be included in the cookbooks. Maybe the evaluation done by the crowd on SO content, already considers the self-containment of the solutions presented, meaning that posts with high score, tend to be self-contained.

The results for *Reprod* were also reasonable as well, indicating that our strategy has good performance in selecting pairs containing snippets that are reproducible or can become reproducible with minor adjusts. The main reasons found, that explains why some source code snippets are difficult to reproduce are:

- The use of a variable that was not declared. For instance, in SWT's cookbook, some answers have snippets that uses a widget object (e.g., a button), but does not show how to create the object. It could not be trivial for someone new to SWT to create these widgets.
- The answerer provide a solution (e.g., a method) and does not show how to use the solution. For instance, some answer snippets present a method as a solution for the question's problem. Sometimes these methods have many parameters, that are not trivial to create. In other words, one of the concerns of the evaluators was to see a working solution, for example, through the use of a "main" method, so they could see, for instance, how to create all objects that a method requires as parameters before calling it.
- The omission of some lines of code (e.g., some answers use "..." to indicate that some lines are omitted in a snippet). For some of these cases, the evaluators were concerned that the lines omitted could be important for the execution of the code snippets.

The two last lines of Table X show the percent of pairs having *Approp*, *SelfCont* and *Reprod* greater than or equal to 3 (seventh row) and equals to 4 (eighth row). More then 65% of all pairs satisfies the condition ($\geq 3$) , and more than 40% of all pair satisfies the condition ($= 4$). These numbers show that our approach has reasonable performance in building cookbooks whose pairs meet *approapriateness, self-containment* and *reproducibility* at least partially.

## C. Threats to Validity

In our approach, the assessment of chapters' semantics is subjective. Different persons may assign different meanings for the same cookbook chapter. Another subjective aspect is the appropriateness of containment of a recipe within a chapter subject. Even considering the same chapter subject, different persons may disagree about that. The evaluation criteria (*Approp*, *SelfCont* and *Reprod*) for a cookbook Q&A pair is also subjective (e.g., one may consider the Q&A pair not suitable for the cookbook, while another person may consider the same pair appropriate). Despite the inherent subjectivity of the evaluation process, we used a double-check strategy aiming to reduce the bias.

Another aspect to note is the overlapping of the issues in the cookbook chapters (i.e., the same issue appearing in more than one chapter of the same cookbook). Although this happens, we conducted a qualitative manual analysis and observed that there were only 2 cases of overlapping in the three cookbooks. Moreover, none of these cases were a complete overlap (i.e., one or more chapters that covers exactly the same issue). For instance, Chapters 3 and 7 of the SWT Cookbook have a partial overlap because both are about the mouse click event. However, Chapter 7 also covers other mouse events, such as drag and drop. One explanation for the overlap is due to the amount of topics found by LDA in the documents. In our approach, the LDA was arbitrarily set to find maximum 20 topics in each cookbook. As our approach is automatic, varying the maximum number of topics to be found by LDA may arise undesirable effects such as greater overlap of issues between chapters or very general chapters. Indeed, there are several thresholds arbitrarily defined. Although, we have pre-tested those thresholds before conducting the whole process, indeed some variation on them may result in different results.

## V. Related Work

We organized the related work in three subsections. In this work, we propose an approach to build a kind of documentation (cookbooks) for APIs, so in Subsection V-A, we review works related to API documentation. The Subsection V-B reviews aspects related to social media in software engineering. In Subsection V-C, we review works related to search of examples to support API documentation and developer's work.

## A. APIs Documentation

The nature of the documentation process differs substantially depending on the context. In the closed documentation, documents rarely leave the environment of a closed system: they are created by few and used by few. Unlike the closed documentation, the API documentation (e.g., Javadocs) is written by a few, but read by many [3]. Documenting APIs effectively and using them is not trivial. Robillard [17] observed that inadequate or inappropriate examples are obstacles for learning APIs. He also identified several other obstacles related to factors such as the format and design aspects of the API. Although useful documentation can be found in Javadoc, the resulting documentation generally narrow in focus and sometimes is too detailed. According to Robillard, the learning process has an API similar to the design of the API importance. He also showed that developers want some complete and well structured documentation with a set of examples showing how to use the API in different scenarios. According to the study participants, the following properties of software documentation are the most important: contents (information in the document), the use of examples, being updated, the organization (index, sections, subsections). In our work, we rely on SO to build the cookbooks and those properties are partially met. The content of the cookbooks are already assessed by the SO community via the voting mechanism. Since SO allows edition of posts (questions and answers), the property of "being update" can be met, as users of SO can edit posts to reflect the changes in new versions of the APIs. As we only selected answers containing source code snippets to compose the cookbooks, the "use of examples" property is also met. The organization property is not always satisfied for the recipes, because not all answers in the cookbooks are organized in sections, subsections. However, the cookbook itself partially meets this property since it is organized into a series of chapters. However, in the current state, there is no coherent sequencing of the proposed cookbook chapters.

## B. *Social Media in Software Engineering*

The possibilities and limitations of the crowd documentation in software development have not yet been fully understood [3]. Parnin and Treude [18] analyzed the results of searches on Google for a specific API (JQuery) and found that, besides the official documentation sources, many sources of documentation via social media appeared among the results. For example, for 84% of methods, they found at least one question on SO on the first page of results.

An analysis of how developers post questions and answers on SO resulted in several categories of questions (how-to, review, error, conceptual, etc.) [7]. Questions containing source code are common in the review category. The study also showed that the SO is particularly effective at code reviews and conceptual questions, and approximately 85% of the questions are answered. Besides the type of question, other factors such as the date and time of posting the question, the identity of the asker, the technology in question, the size of the question, and the presence of source code in question affect its likelihood of receiving good responses. In their study of the design elements behind the SO, Mamykina et al. [19] have concluded that its success is not explained only by the technical design of the site, but also for the assessment activities (e.g., voting) and incentives (such as the reputation score). They also found that users of Stack Overflow get very quick answers: the questions are answered in about 11 minutes.

Our work is also related to social media because it uses the "crowd knowledge" available in SO to build cookbooks for APIs. As we assessed a series of criteria for the generated cookbooks, this work contributes to understand how the social media can be used in software documentation field.

## C. *Use of Examples*

The use of examples is a important source of help for programmers because they can solve a problem through reuse of examples [20]. There are tools designated to extract code from open source repositories to identify examples. Some tools extract common patterns of use of APIs [21] [22]. Other approaches, use data mining techniques to locate characteristics in the source code [23]. Those tools rely on queries consisting of keywords to find examples. Holmes et al [24] developed a tool (Strathcona) that uses the current context of development (e.g., the classes being used in the source code) as a query to find similar source code snippets to be reused. Another examples and artifacts recommendation tool is Hipikat [25]. In the work presented here, we also need to select good examples that are inserted into the chapters of the created cookbook. The difference with respect to these works is that in our approach, the examples of API usage are mined from SO (in contrast with repositories of source code). Moreover, as a cookbook is a documentation for a whole topic (e.g., SWT), our approach is characterized as a browse-oriented interface instead of a search-oriented interface, i.e., we recommend a entire set of examples and explanations instead of focusing on examples for a particular problem that a user has at hand.

Stylos et al. [26] developed a tool (Jadeite) that extracts some common usage scenarios and inserts it into the API documentation (Javadoc), however the examples are only limited to instantiation of classes. Our work is related to cookbooks instead of Javadocs. Moreover, the cookbooks are created from the scratch (i.e., we are not improving an existing documentation).

Ponzanelli et al. [27] presented an integrated and largely automated approach to assist programmers 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. Our approach also uses data from SO, but has a browsing interface in contrast to SEAHAWK's query interface. Moreover, instead of a plugin for a IDE, we developed a website where the cookbooks are available.

# VI. Conclusions

In this paper, we presented an approach to build cookbooks (a recipe-oriented documentation) for APIs. The information used to build the cookbooks is mined from SO, a Question and Answer website related to software development. The information available in this kind of social media services is often called "crowd knowledge".

Our approach aimed at addressing the problem of the lack of documentation or poor quality documentation for APIs using the information available in SO to build cookbooks. In the process of building the cookbooks, we organized the information available in SO about a certain API into a series of chapters, each one related to a subject or theme of the API. To identify those subjects, we used LDA, a topic modeling technique. Each topic found by LDA was considered to be a potential chapter in the cookbook. The chapters were filled with recipes, which are composed by a question and one or two answers to it. We aimed at selecting only one special kind of question to be included in cookbooks: the *How-To-Do-It* question, that corresponds to questions asking for instructions on how to solve a task using a API. A simple, yet adequate, rule-based approach was developed to filter *How-To-Do-It* question. Moreover, we selected only well voted posts from SO to be included in the cookbooks.

The evaluation of our cookbook generating approach was conducted with three important APIs: SWT, STL and LINQ. These APIs are widely used by the software development community. Our evaluation was targeted at evaluating the quality of the proposed organization and of the selected recipes. We did not evaluate the comprehensiveness of cookbooks, because this criteria would be intrinsically linked with the comprehensiveness of "crowd knowledge" around the desired API. We defined some criteria to test the quality of the generated cookbooks: we checked whether the cookbook chapters had well defined meaning; if the recipes were related to the meaning of its chapter; and how the question and answer pairs presented in the cookbooks met three properties (the appropriateness to be in a cookbook; the self-containment of its information and the reproducibility of its source code snippets). The results of the evaluations were promising: 63–71% of the chapters proved to have a defined meaning; 72–88% of the recipes are related to the meaning of the chapter that it belong; more than 65–75% of the pairs presented in the cookbooks meet, at least partially, the properties of appropriateness to be part of a cookbook, self-containment and reproducibility of snippets. We developed a website were the cookbooks built for the three APIs can be seen.

As a future work, we intend to apply the proposed approach to generate cookbooks for others APIs. We also expect to define other criteria to evaluate other dimension of the quality of cookbooks. There are still some problems that should be investigated, such as, organizing the order of the chapters in a coherent way, or even to propose different organizations, such as tutorials or books.

## Acknowledgments

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

## References

[1] F. P. Brooks, Jr., *The Mythical Man-month (Anniversary Ed.).* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[2] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What should developers be aware of? An empirical study on the directives of API documentation." *Emp. Softw. Eng.*, vol. 17, no. 6, pp. 703–737, springer US, 2012.

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

[4] D. Kramer, "API documentation from source code comments: a case study of javadoc," in *Proc. of the 17th annual Int. Conf. on Computer Documentation*. New York, NY, USA: ACM, 1999, pp. 147–153.

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

[6] A. Mockus, R. T. Fielding, and J. Herbsleb, "A case study of open source software development: the Apache server," in *Proc. of the 22nd ICSE*. ACM, 2000, pp. 263–272.

[7] C. Treude, O. Barzilay, and M.-A. Storey, "How Do Programmers Ask and Answer Questions on the Web? (NIER track)," in *Proc. of the 33rd ICSE*. ACM, 2011, pp. 804–807.

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

[9] S. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What Makes a Good Code Example? A Study of Programming Q&A in Stack Overflow," in *Proceedings of the 28th IEEE ICSM*, 2012, pp. 25–34.

[10] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of machine Learning research*, vol. 3, pp. 993–1022, published by JMLR. org, 2003.

[11] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in Stack Overflow," *Empirical Software Engineering*, pp. 1–36, springer US, 2012.

[12] S. W. Thomas, "Mining software repositories using topic models," in *Proceedings of the 33rd ICSE*. New York, NY, USA: ACM, 2011, pp. 1138–1139.

[13] M. F. Porter, "Readings in Information Retrieval," K. Sparck Jones and P. Willett, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, ch. An algorithm for suffix stripping, pp. 313–316.

[14] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno, "Evaluation methods for topic models," in *Proceedings of the 26th Annual ICML*. New York, NY, USA: ACM, 2009, pp. 1105–1112.

[15] T. L. Griffiths and M. Steyvers, "Finding Scientific Topics," *PNAS*, vol. 101, no. suppl. 1, pp. 5228–5235, 2004.

[16] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A Study of the Difficulties of Novice Programmers," *SIGCSE Bull.*, vol. 37, no. 3, pp. 14–18, aCM, 2005.

[17] M. P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," *IEEE Softw.*, vol. 26, no. 6, pp. 27–34, iEEE Computer Society Press, 2009.

[18] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proc. of the 2nd Web2SE*. ACM, 2011, pp. 25–30.

[19] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest Q&A site in the west," in *Proceedings of CHI*. New York, NY, USA: ACM, 2011, pp. 2857–2866.

[20] M. B. Rosson and J. M. Carroll, "The reuse of uses in Smalltalk programming," *ACM Trans. Comput.-Hum. Interact.*, vol. 3, no. 3, pp. 219–253, aCM, 1996.

[21] M. Acharya, T. Xie, J. Pei, and J. Xu, "Mining API patterns as partial orders from source code: from usage scenarios to specifications," in *Proceedings of the the 6th joint meeting of the European software engineering conference*. ACM, 2007, pp. 25–34.

[22] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in *Proc. of the 18th ACM SIGSOFT FSE*. ACM, 2010, pp. 157–166.

[23] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proc. of the 33rd ICSE*. ACM, 2011, pp. 111–120.

[24] R. Holmes and G. C. Murphy, "Using structural context to recommend source code examples," in *Proceedings of the 27th ICSE*. ACM, 2005, pp. 117–125.

[25] D. Čubranić and G. C. Murphy, "Hipikat: recommending pertinent software development artifacts," in *Proceedings of the 25th ICSE*. IEEE Computer Society, 2003, pp. 408–418.

[26] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving API documentation using usage information." in *CHI Extended Abstracts*. ACM, 2009, pp. 4429–4434.

[27] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging Crowd Knowledge for Software Comprehension and Development," in *CSMR*, A. Cleve, F. Ricca, and M. Cerioli, Eds. IEEE Computer Society, 2013, pp. 57–66.