Leveraging Crowd-sourced Information to Guide Library Usage

by

Benyamin Noori

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Studies have shown that there is a mismatch between the information needs of a developer and information provided by the documentation of software libraries. Additionally, previous work suggests that developers' information needs are driven by the tasks they do. Based on these previous findings, we argue library documentation should be task-oriented. We propose a technique to generate task-oriented library documentation based on information extracted from the popular crowd-sourced Q & A website, Stack Overflow. Our methodology makes use of natural language processing techniques as well domain-specific heuristics to process and organize information available on Stack Overflow. The resulting task-oriented documentation for a software library contains three main components: (1) a list of tasks that can be achieved using the library, (2) a set of code snippets to demonstrate how to accomplish a specific task, and (3) additional information about trade-offs and insights of using specific Application Programming Interfaces.

To evaluate the quality of information we extract and the usefulness of our approach, we conduct a survey of a diverse group of 69 Java developers. In addition to showing that our proposed task-oriented library documentation is useful to developers, the findings of our online survey also shed light on current limitations and challenges of extracting information from crowd-sourced websites. Finally, we present improvements to parts of our methodology, based on our analysis of the results.

Preface

Chapter 8 is the result of an international research collaboration with Dr. Christoph Treude from the University of Adelaide.

The research project, of which this thesis is a part, received research ethics approval from the University of Alberta Research Ethics Board, Project Name "Task-based Code Recommender Systems", No. REQ00001293, approved on 08/06/2017.

Run, rabbit run.
Dig that hole, forget the sun.
When at last, the work is done,
Don't sit down, it's time to dig another one.

– Roger Waters, 1973.

Acknowledgements

I would like to thank my supervisor, Dr. Sarah Nadi. Her continous guidance and support along with constructive feedback and criticism made this work possible.

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Library documentation contains the information needed by a developer to use the given library and its corresponding Application Programming Interface (API). However, the mismatch between information needs of a developer and what is provided in documentation has led to observations of developers struggling to find accurate and easily consumable information at the right time [21], [35], [50].

While there has been several research efforts to improve software documentation for developers [8], [14], [41], [46], [47], studies have shown that not only is the quality of the documentation important, the aggregation of information presented should match the way developers think about everyday software development activities [34]. Specifically, previous work suggests that concrete software development tasks often drive developers' information needs [28], [36], [47].

Given the findings of previous work, we argue that a library's documentation should be driven by tasks that developers want to accomplish. In our work, we define a *development task*, *task* for short, to be a specific programming action, such as *"Create a file and write to it"*, *"Convert a string to a JSONObject"*, or *"Connect to a database"*.

For a given software library, we propose creating a task-oriented documentation that consists of three components: (1) a number of software development tasks that can be accomplished using the library, (2) a set of code snippets demonstrating various ways of accomplishing the task, and (3) in-

sight sentences that discuss nuances of different approaches. To create such documentation, we leverage the rich information found on the crowd-sourced question-answer website, Stack Overflow. Stack Overflow has become a popular online resource where developers find solutions to their programming questions. It currently hosts more than 100 million posts [39].

Software developers use Stack Overflow in order to find ways to accomplish everyday development tasks. To show the type of information available on Stack Overflow, we use a scenario where a developer intends to open a file and write some information to it. Using the search capability on Stack Overflow, they might come across the thread displayed in Figure 1.1.

As we can see, the first answer posted to this question suggests multiple solutions that can accomplish the task in question. We can see that two concrete code snippets are provided in the answer. The writer also adds more detail to their answer by saying *"Note that each of the code samples below throw IOExcepions"*.

Other solutions posted in this thread propose a variety of different solutions, each of which are associated with certain advantages and disadvantages. In fact, a simple search on Stack Overflow will return many threads about the same task or similar ones.

Such information can help the developer make better decisions in terms of design and implementation. However, the amount of information a developer needs to go through on Stack Overflow can be daunting. Task-oriented documentation provides all the information necessary to complete a software development tasks all in one place and it does so in a way that's useful for software developers. This type of documentation technique filters, aggregates and reorganizes relevant information from Stack Overflow for a given library.

Specifically, our task-oriented library documentation shows a list of tasks that can be accomplished with the library, as shown in Figure 1.2 for *junit*. When the user clicks on one task, they would then see the task documentation page in Figure 1.3, which consists of the three components mentioned earlier.

To create a task-oriented documentation page of a library, we build on and extend existing techniques [46], [47]. We also make use of Natural Language

Figure 1.1: Stack Overflow thread number 2885173

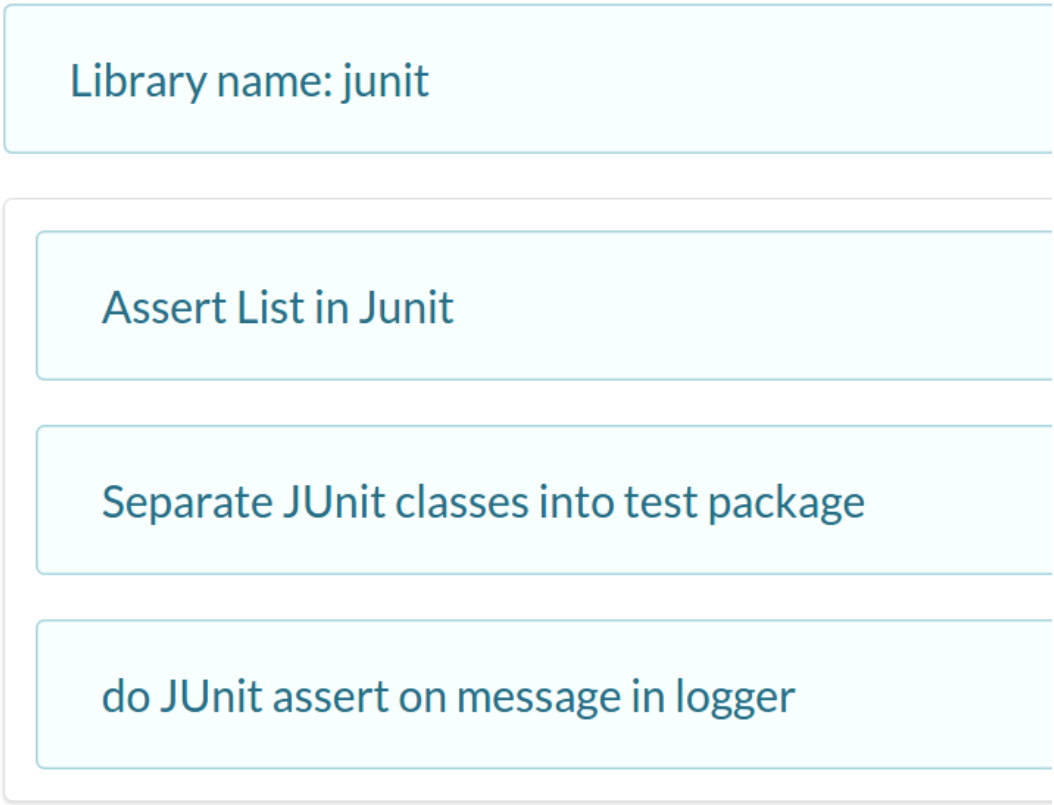

Figure 1.2: A list of tasks for a given library.

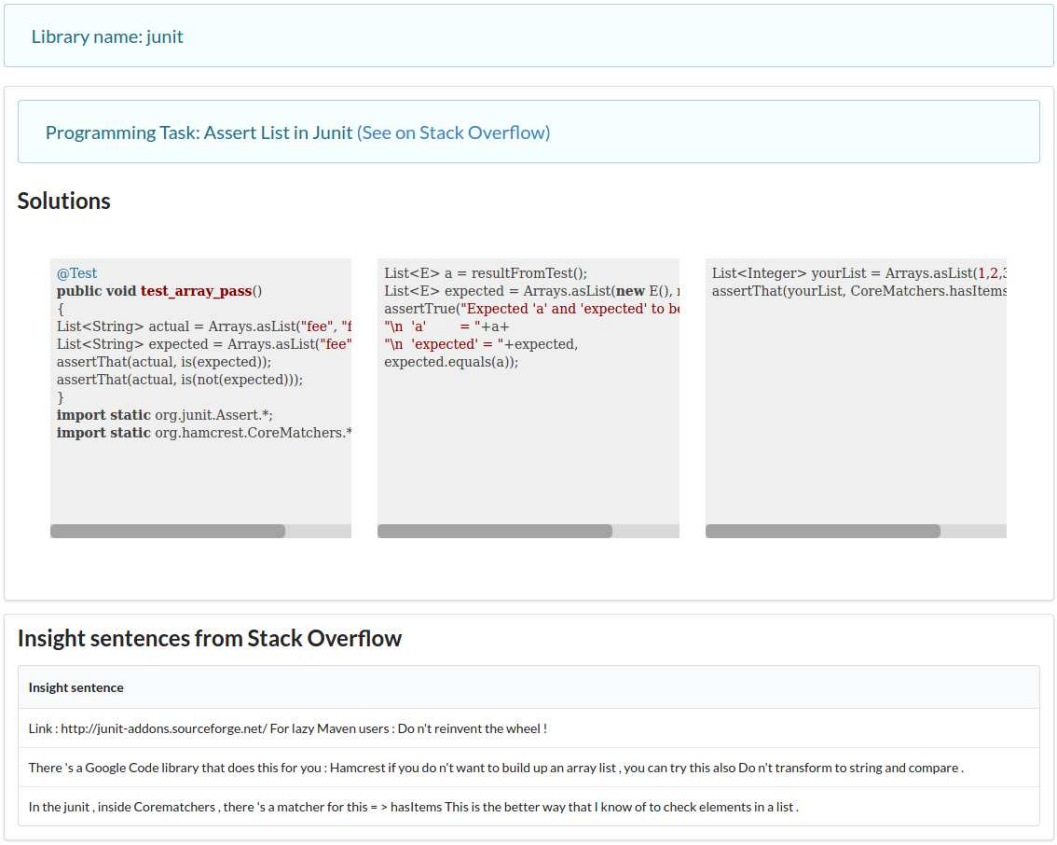

Figure 1.3: A sample task documentation.

Processing (NLP) techniques as well as heuristics specific to the domain of our study. To evaluate our proposed documentation, we survey 69 Java developers, asking them to evaluate the quality and the usefulness of the information extracted using our approach.

Our results show that software developers find the idea of task-oriented documentation useful. The ratings of the usefulness and quality of the three components of the survey varied, which allowed us to identify potential improvements for further pushing documentation extraction techniques. We provide an extensive discussion of the challenges we identified and suggestions for addressing them.

Based on the results of our experiment, we also propose and implement an improved insight sentence extraction methodology that specifically focuses on conditional insight sentences. We define conditional insight sentences as sentences in Stack Overflow text that recommend a strategy or solution based on a certain condition.

To summarize, this thesis makes the following contributions:

1. We introduce a methodology to build task-oriented documentation of software libraries from Stack Overflow data by adapting and combining several existing techniques. Our toolchain is publicly available and can be adapted to other documentation sources [49].

2. We demonstrate the usefulness of the proposed documentation technique through a survey of 69 Java developers with diverse backgrounds.

3. Based on our results, we propose and implement an improved methodology to extract insight sentences. We also provide a discussion on how to further improve task-oriented documentation.

The rest of the thesis is organized as follows. In Chapter 2, we review the related literature. Chapter 3 describes our methodology to build task-oriented library documentation. Chapters 4 and 5 describe the developer survey and the results, respectively. Chapter 6 provides an in-depth discussion and analysis of the results. Chapter 7 discusses the threats to the validity of the results

presented in the thesis, Chapter 8 describes our improved methodology to extract insight sentences and Chapter 9 concludes this thesis.

Chapter 2

Literature Review

In this chapter, we review studies on library documentation and Stack Overflow data. We partition the existing literature into three parts: (1) Assisting software developers in using software libraies, (2) Augmenting library and Application Programming Interface (API) documentation, and (3) Tools and methodologies using Stack Overflow data.

2.1 Assisting Software Developers in Using Software Libraries

Uddin et al. [51] introduce OPINER, an online opinion summarization engine that generates summaries of reviews based on sentiment analysis, natural language processing as well domain-specific knowledge. de la Mora and Nadi present a metric-based comparison of software libraries based on a variety of information available about a library, such as the number of issues and frequency of releases [26]. While these efforts focus on helping software developers select libraries, our work intends to create documentation of a given library and not provide information comparing two or more libraries.

Others have introduced methodologies and tools to present developers with code snippets and examples from various sources in order to help developer use APIs. `Mica` is a search engine, proposed by Stylos and Myers [40], that identifies examples of API usages through filtering query results returned from a standard search engine such as Google. After obtaining a set of search results, it analyzes the contents of those results to find relevant programming

terms and classifies each result as either method or class. `ROSF` by Jiang et al. is a similar work whose goal is to search the web for code snippets given a query [19]. Unlike previous methodologies that only relied on information retrieval techniques to find search results, `ROSF` combines both information retrieval and supervised learning techniques.

Moreno et al. introduce `MUSE` (Method USage Examples), a technique for mining and ranking code examples that demonstrate how to use a specific method [27]. They accomplish this using static slicing with clone detection, and heuristics to select and rank the best examples in terms of reusability, understandability, and popularity.

Thung introduces a new approach to recommend APIs to developers at different granularity levels based on the development task at hand [43]. The development task at hand in this work is defined a textual description of a new functionality or feature that needs to be implemented by a software developer. This recommendation takes place in multiple stages: (1) recommending libraries to use [44], (2) recommending API methods to use [45], (3) recommending the parameters to be used and (4) recommending a combination of API methods to accomplish a given task description.

Additional work integrates the search or presentation of API usage examples into the developers' Integrated Development Environment (IDE). `eMoose` is an IDE plugin created by Dekel and Herbsleb [14] that highlights API usages that have directives associated with them in the documentation. Campbell and Treude [6] created an IDE plugin that can import code directly from Stack Overflow given a natural language description of what the developer wants to do. A similar work by Ponzanelli et al., `Prompter`, can retrieve the related Stack Overflow discussions given the development task at hand [33]. Similarly, `seahawk`, introduced by Ponzanelli et al.[32], is a plugin that automatically looks for related threads on Stack Overflow according to the context of the IDE. Even though we also rely on Stack Overflow, our methodology and goals are different since we do not use the developer's current code context to search for information. Instead, we focus on creating documentation that may be useful to several developers.

The main difference between this thesis and the work described in this section is that in addition to code snippets and code examples, our methodology also provides insight sentences, important text snippets from Stack Overflow. Also, we organize information around software development tasks to create documentations of libraries.

2.2 Augmenting library and application programming interface (API) documentation

There is also extended literature on the subject of improving the documentation of software development technologies to make them more useful for software developers. Treude et al. proposed an approach to extract software development tasks from API documentation text [47]. We use their proposed methodology for identifying tasks, but instead of focusing on existing official library documentation, we extract tasks from crowd-sourced information in Stack Overflow threads. Treude and Robillard introduced a supervised approach to identify insights from Stack Overflow and add them to an API's documentation[46]. We modify their methodology for identifying insights to adapt it to the context of our task-oriented documentation page. Subramanian et al. [42] introduced a methodology to link source code examples from Stack Overflow to API documentation, and therefore enhance the documentation. Chen and Zhang [8] proposed integrating frequently asked questions on crowd-sourcing websites into the documentation in order to give more convenient access to the information sought by the developer. Stylos et al. developed `Jadebite`, a tool that utilizes API usage statistics to build a Javadoc-like documentation by marking popular API elements [41]. While all the above techniques can complement our work and we build on some of them, our goal is to create a standalone documentation page that can easily be consumed by software developers looking to perform a given task. In their paper *"On Demand Developer Documentation"*, Robillard et al. describe a vision of an on-demand documentation system that can automatically generate high-quality documentation in response to a user query. Such a system would use knowl-

edge extraction techniques on structured and unstructured data available e.g. source code, issue tracking system metadata, and posts from Q & A discussions. The information gathered by our task-oriented documentation can be used in such an on-demand developer documentation system.

The literature in this area mainly focuses on ways of improving information provided in the documentation. Our work is different given that we intend to build a task-oriented documentation organized around a list of software development tasks which is different from regular documentation available for software libraries while these examples have been designed to add information to documentation to make it more useful for software developers.

2.3 Tools and methodologies using Stack Overflow data

Some of the work we have mentioned so far uses Stack Overflow as a main source of data for various purposes [6], [8], [32], [33], [42], [46]. There has also been work that uses Stack Overflow for slightly tangential purposes. For example, Barua et al. applied machine learning techniques to discover a set of topics discussed by developers on Stack Overflow [1]. Parnin et al. examined crowd sourced documentation, i.e. discussions on Stack Overflow in terms of coverage, dynamics, and quality [29].

However, our work is different from such related work, due to the fact that we use this data to specifically create task-oriented library documentation to the best of our knowledge there has been no effort to achieve this before.

Chapter 3

Generating Task-oriented Library Documentation

In this section, we describe the details of our approach. We begin by selecting a target library and gathering data relevant to the target library from Stack Overflow and proceed to process that information and organize it in a format that allows us to create our task-oriented documentation. A general view of the desired documentation can be seen in Figure 1.3. A general overview of our methodology can be seen in Figure 3.1.

What follows is a step-by-step description of the steps in our methodology. To better demonstrate the processing done in each step, we use the example of *junit* [20], a widely used testing framework for Java. We describe the results of each step with *junit* as our target library.

3.1 Acquisition of Relevant Threads from Stack Overflow

Questions on Stack Overflow are always associated with a number of tags. A *tag* is a word or phrase that describes the topic of the question. Tags are a means of connecting experts with questions they will be able to answer by sorting questions into specific, well-defined categories [53]. Each library often has a corresponding tag on Stack Overflow. For instance, `Junit`'s tag is *junit*. We use tags to identify all the questions on Stack Overflow that are relevant to a given library.

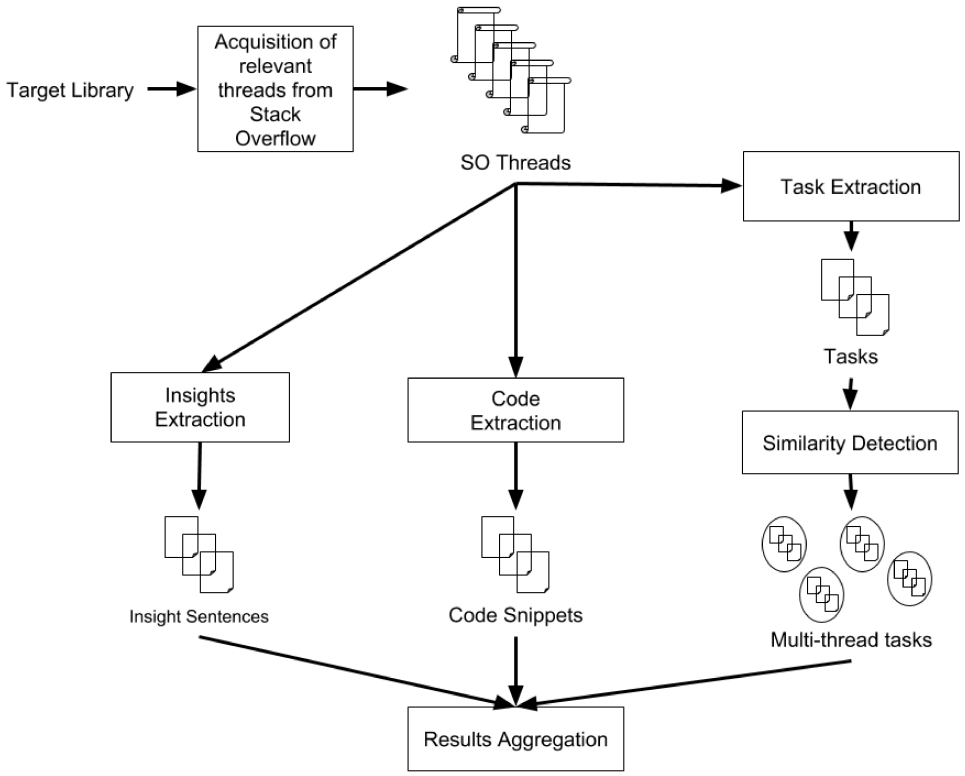

Figure 3.1: An overview of our methodology.

For our methodology, we assume the tag associated with a library is known. We begin by manually identifying the corresponding tag of the target library. This can be accomplished easily through a search on Stack Overflow. We then use the Stack Exchange Data Explorer [39], an online tool that allows users to perform SQL-like queries over a virtual view of Stack Overflow's database, to collect the data for all Stack Overflow threads related to a given library, i.e. questions tagged with the library's specific tag.

For every question tagged with the library's tag on Stack Overflow, we collect the question title, question body, question score, tags associated with the question, answers posted in the thread, and the ID of the accepted answer, if any. For every answer, we collect the ID of the answer, its score, and its body text. In the case of `Junit`, we find 45,644 questions on Stack Overflow associated with the *junit* tag. We collect the information mentioned above for each of these questions.

3.2 Task Extraction

There are a variety of questions on Stack Overflow. For example, users can discuss why a certain runtime exception is thrown or what a certain concept means and where it is used. They can also ask questions about how to accomplish everyday programming tasks. Given the focus of our work, we need to distinguish the latter type of questions, referred to from now on as *task questions*, from the rest of the threads collected from Stack Overflow.

We use natural language processing to differentiate task questions from other types of questions. We follow a methodology presented by Treude et al. [47] for identifying sentences that represent tasks. The original goal of Treude et al.'s work is to identify sentences in technology documentation that represent software development tasks. While we want to identify tasks in Stack Overflow threads, their methodology applies to this problem as well. This methodology heavily relies on the functionality provided by the Stanford CoreNLP Toolkit [22]. Specifically, we use the *part-of-speech tagger*, a tool that determines the grammatical roles of words in a sentence and the *dependency parser*, a tool that determines the grammatical relationship between words in a sentence.

By reading through a sample of Stack Overflow data, we observe that users on Stack Overflow tend to describe the programming tasks in the titles of their questions and add context and more details in the body of the question. This added context might include providing in-depth descriptions of issues into details specific to their case or adding code snippets they are working with. Given our goal of identifying whether a given question is a task question or not, we focus on the titles of Stack Overflow questions and identify those that represent tasks through a two-phase processing: (1) Identification and (2) Extraction. The first phase determines whether the question is related to a task. The second phase then takes the title of each thread as input, and creates a sentence describing the programming task. We now describe the details of each phase. An overview of the task extraction process can be seen in Figure 3.2.

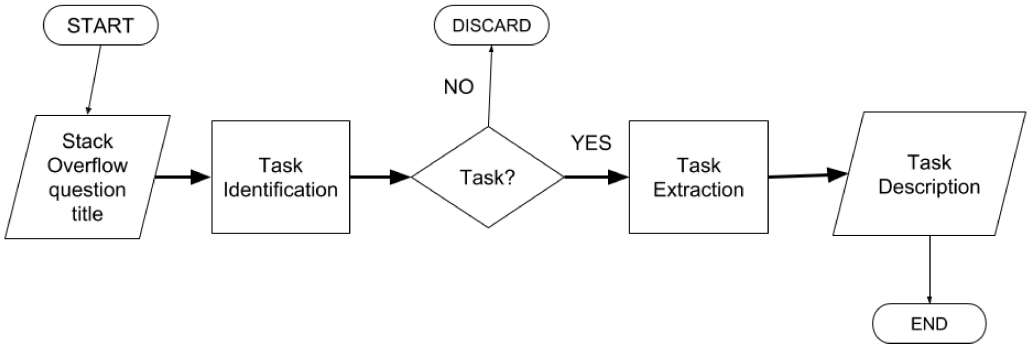

Figure 3.2: Overview of task extraction process.

3.2.1 Task Identification

In Phase 1, we use a set of grammar dependencies for identifying tasks, created by Treude et al. [47]. A list of these grammatical dependencies can be seen in Table 3.1. They specify that a sentence is a task if two words in that sentence are dependent with one of these dependencies. One such dependency is the direct object relationship. Direct object is the relationship between a verb and its direct object. For example in the following sentence, the words *receipt* and *generate* are dependant with a direct object relationship: *"This can be used to generate a receipt."* A set of examples can be seen in Table 3.2

We feed every question title from Stack Overflow to CoreNLP's dependency parser. We look through the dependencies to find sentences that contain any of the predefined dependencies that represent a task. If such a dependency exists among the words in the thread title, that thread is labelled as a task question, and not otherwise.

In this phase, question titles such as *"How do you assert that a certain exception is thrown in JUnit 4 tests?"*, *"How to run test methods in specific order in JUnit4?"*, and *"Conditionally ignoring tests in JUnit 4?"* would be labelled as task questions. Titles like *"Why doesn't JUnit provide assertNotEquals methods?"*, *"Difference between setUp() and setUpBeforeClass()"* would be labelled as non-task questions.

Dependency	Description
Direct object (*dobj*)	The noun phrase which is the (accusative) object of the verb.
Prepositional modifier (*prep*)	Any prepositional phrase that serves to modify the meaning of the verb, adjective, noun, or even another preposition.
Agent (*agent*)	The complement of a passive verb which is introduced by the preposition "by" and does the action.
Passive nominal subject (*nsubjpass*)	A noun phrase which is the syntactic subject of a passive clause.
Relative clause modifier (*rcmod*)	A relative clause modifying the noun phrase.
Negation modifier (*neg*)	The relation between a negation word and the word it modifies.
Phrasal verb particle (*prt*)	Identifies a phrasal verb, and holds between the verb and its particle.
Noun compound modifier (*nn*)	Any noun that serves to modify the head noun.
Adjectival modifier (*amod*)	Any adjectival phrase that serves to modify the meaning of the noun phrase.

Table 3.1: List of dependencies from Treude et al., 2015 [47] used for task identification.

3.2.2 Task Extraction

Only task threads identified Phase 1 would proceed to Phase 2. The goal of Phase 2 is to transform a question title in to a sentence describing a programming task. To do so, we feed the sentence in to Core NLP's part-of-speech tagger. Next, we select all words that either depend on a verb or that a verb depends on, as well as the verbs themselves in the same order in which they appear in the sentence. We also remove stop words and stem the verb in the sentence to help with identifying similar items in a later step. The resulting sentence would be one describing a programming task. For example, the question titles *"How do you assert that a certain exception is thrown in JUnit 4 tests?"*, *"How to run test methods in specific order in JUnit4?"*, and *"Conditionally ignoring tests in JUnit 4?"* would be transformed to the following task descriptions *"Assert that certain exception is thrown in Junit 4 tests"*,

Dependency	Example
dobj	This can be used to *generate* a *receipt* or some other confirmation.
nsubjpass	The thumbnail *size* is *set* in your templates.
rcmod	It allows you to set one *rate* that is *multiplied* by the number of items in your order.
prep	There are a couple of different ways to *integrate* with Google *Checkout*.

Table 3.2: Examples of grammatical dependencies used for task identification.

"Run test methods in specific order in JUnit 4", and *"Conditionally ignore tests in JUnit 4"*, respectively. We then associate this task description with the original question for future processing.

3.2.3 Evaluation

In order to evaluate the accuracy of the task identification strategy over Stack Overflow data, we prepared a set of 200 randomly selected question titles. We labelled this set manually with two classes: positive or negative. In order to label this set, we follow a definition by Treude et al.[47] which defines a software development task as "a specific programming action". After the labelling of the set was complete, we applied the task identification process described here. We obtained an accuracy of 74.5%, a precision of 85.9%, and a recall of 79.7%.

We also used this data set to investigate the number of times each dependency listed in Table 3.1 is found in the task extraction process. We can see from Table 3.3 that "dobj" is the most common types of dependency. Note than one question title from Stack Overflow can be associated with multiple dependency types.

3.3 Similarity Detection

One of the main challenges of reading through threads on Stack Overflow is that several threads discussing the same software development task may exist, and it is often infeasible for a user to study all of that content. We can address this by aggregating data from similar threads. In order to identify

Dependency	Number of Observations
dobj	131
rcmod	119
agent	117
amod	44
nsubjpass	13
neg	8
prep	5
nn	5
prt	1

Table 3.3: Frequency of grammatical dependencies in a set of 200 randomly selected question titles.

similar threads, we find threads that have similar task descriptions associated with them.

We first model each task sentence as a vector of its terms. Each element of the vector represents one word and the value of each element shows the frequency of that word in the sentence represented by the vector. The similarity of two tasks is then calculated as the cosine of the angle between the two vectors representing the task descriptions.

Essentially, the similarity is equal to $cos(\theta)$ where θ is the angle between the two vectors, as indicated in Equation 3.1:

$$similarity = cos(\theta) = \frac{A \cdot B}{||A||||B||} \tag{3.1}$$

where A and B are the vectors representing the task descriptions of the two given questions. If the similarity score of two tasks is higher than a predetermined threshold, the two tasks are deemed similar.

To determine the proper threshold for similarity, we prepared a set of 55 manually labelled samples to test the thresholds with. We then tested 5 different threshold values on our test set. The values we used were 0.3, 0.5, 0.6, 0.7, 0.9. Based on the accuracy obtained using each value, we chose a threshold of 0.6.

One issue with cosine similarity is that sentences with many similar terms can have very different meanings. For instance the following sentences have

exactly the same set of terms but different meanings: (1) *Convert string to int in java* and (2) *Convert int to string in java*. With cosine similarity, these two sentences will have a similarity score of 1. We address this problem by adding shingling. *Shingling* is a widely used technique in information retrieval [4] where instead of modeling a sentence as a set of its terms, it is modelled as a set of phrases of size x, where x is some determined window size that slides across the sentence. What is seen through that window at each step is one item in the set of terms that represent a sentence.

For example, with $x = 2$, sentence 1 from the above examples would be represented with the following set of terms: { *"Convert string"*, *"string to"*, *"to int"*, *"int in"*, *"in java"*}. In our approach, we apply shingling with a window of size 2 and then compare the similarity of the sets of 2-word terms for task descriptions extracted from Stack Overflow threads. We chose a window size of 2 because the task descriptions are often short and a larger window size would create lengthy and very unique phrases which would negatively impact the similarity extraction. On a high level, shingling allows for a level of consideration for word order by considering more meaningful phrases to determine sentence similarity.

Finally, based on pairs of similar tasks, we can create clusters of tasks that are similar to one another. There can be clusters that include only one task (i.e. ones that do not have any similar tasks), and ones with multiple tasks. We group the tasks in each cluster and create a number of multi-thread tasks. A *multi-thread tasks* includes a number of tasks extracted from multiple threads, all of whom are similar. These tasks, just like any other task in our work, have a task description and a set of answers. The following is a description of how we find the task description and answers of a multi-thread task.

We first find the task associated with a question that has the highest score among all in the multi-thread task. This task will represent the multi-thread task. The set of answers for a multi-thread task is obtained by grouping all the answers from all the tasks in the corresponding cluster and sorting them based on their corresponding question score on Stack Overflow.

In short, for every set of similar tasks, we aggregate all their answers. We

use the task description from the question with the highest score, as the task description of this new aggregated task. Obviously, there can be threads with no similar threads in the dataset. These threads will remain the same after this step.

Note that while each multi-thread task is represented by only one development task, it contains information, such as code snippets and answers, from multiple Stack Overflow threads, which we use to construct our documentation for that task.

3.4 Code Extraction

The next step includes processing the answers in our set of multi-thread tasks to find solutions that demonstrate how a developer can accomplish those tasks. We focus on solutions that are in the form of code snippets. There are two types of code snippets in Stack Overflow threads:

1. *Inline code snippets*: These snippets often mention function names or short commands. Inline code snippets are descendants of HTML paragraphs (`<p>` HTML tags) and are embedded in `<code>` HTML tags.

2. *Non-inline code snippets*: These often show a block of code that demonstrates how to accomplish a certain task.

   For our purposes, we want to extract the latter type of code snippets. Non-inline snippets are descendants of pre-formatted HTML blocks (`<pre>` HTML tags) and are embedded in `<code>` HTML tags.

We take the non-inline code snippets found in the answers of all similar threads combined and sort them according to the score of their corresponding answer on Stack Overflow. In case of a tie, we give priority to the answer that has been marked "Accepted" by the asker of the question. Note that this list of snippets has been compiled from multiple similar threads about the same development task. Given that we sorted answers based on their scores on the website, the code snippets will also be sorted based on the score of their corresponding answer.

3.5 Insight Extraction

This process has two steps. 1) Classification: In this step we classify a given sentence as either an insight or not an insight. 2) Outside dependency: We check to see if the sentence has any outside dependency. The goal of the second step is to determine whether the sentence makes sense outside of its context.

3.5.1 Insights Classification

Another important component of our proposed library documentation are insights, a concept originally introduced by Treude and Robillard [46]. The goal of their work was to augment API documentation by dynamically adding insights gathered from Stack Overflow.

According to their work, developers on Stack Overflow discuss information not contained in the documentation. These discussions can include alternative solutions to a programming task. They may also discuss in-depth information about a certain class, software library, or API. Given the goal of their work, they define insight sentences as those sentences on Stack Overflow that are related to a particular API type and that provide insights not contained in the API documentation of the type. Thus, their extraction of insights depends on comparing sentences from Stack Overflow to sentences in the documentation.

Due to the fact that we do not consider API documentation in the scope of our work, we use a variation of this definition as follows: an *insight sentence* is one that falls in at least one of these categories:

1. Discusses general points, advice, or extra information about a task.
2. Discusses APIs that can be used for a task.
3. Offers in-depth information about an API used to accomplish the task.
4. Offers information about alternative solutions/APIs, their advantages or disadvantages.

In order to identify the insight sentences from the bulk of text from the collected answers, we train a supervised classifier over a set of manually labelled

sentences gathered from Stack Overflow answers. We randomly selected a set of 495 sentences from the text of the answers in our dataset. Two contributors independently labelled the sample and then resolved any conflicts. Out of 495 sentences, there were 22 conflicts in the labeling. We calculated the Kappa score for our labelling of this dataset and obtained a score of 0.897. With this approach, we were able to build a ground truth set for 495 sentences with 143 positive labels and 352 negative labels.

We can observe that the majority of the data points (71.11%) of the ground truth data sample have negative labels. This imbalance in the dataset impacts our choice of performance measure. If we assume the ratio of positive to negative data points will hold for unseen data samples, a classifier that always labels its input as negative will have an accuracy of 71.11%. This example demonstrates that accuracy is not a good reflection of classifier performance here, which drove us to look for a different performance measure to evaluate the classifier. Given the imbalance in our dataset, we use the F-1 score, which is the harmonic mean of precision and recall, to properly reflect the performance of each classifier.

In their work [46], Treude and Robillard use a variety of features to build a supervised classifier to identify insights. Table 3.4 holds a summary of the features they used.

Features that fall in the similarity category are not applicable to our work, given that we do not want to compare against existing documentation of a library. We built supervised classifiers using various combinations of the remaining features presented in Table 3.4. In the end, it became clear that the classifier we built using only words as features (the first item in the table of feature sets) has the best performance score, therefore we opted to not use features other than the words in the sentence. In our dataset, each sentence is modelled as a vector where each element represents one word in the vocabulary of the dataset. The value of each element represents whether the term has appeared in that sentence or not. We use these vector representations of sentences for training and testing the classifier.

We built several classifiers using various machine learning models. We

Feature Set	Description
Sentence	Features obtained from the sentence such as the set of words, part-of-speech tags of words, the number of words in the sentence, percentage of tokens tagged with HTML tags, etc. These features can be obtained through parsing the text as well as using CoreNLP's part-of-speech tagger.
Question	This group has features like the question score, whether the question includes an API element name, the asker's reputation, etc. This information can be obtained from Stack Overflow as well as by parsing the data.
Answer	Features such as the score of the answer, the reputation of the asker, whether the answer was accepted, and the number of stars the answer has received fall in this category of features.
Similarity	This set of features is specific to the work presented in [46]. These features include items like average cosine of similarity of the sentence with sentences of the documentation. In order to obtain these features we need to parse sentences of the documentation in question and compare each sentence in the data to each sentence in the documentation to calculate similarity.

Table 3.4: Features used in the work of Treude and Robillard [47].

use five-fold cross validation to evaluate these classifiers. Table 3.7 shows a summary of our results. We obtained the best performance using an SVM model with stochastic gradient descent learning with a F-1 score of 0.714. In order to train the model, we use NLTK's Stochastic Gradient Descent Classifier (*sklearn.linearmodel.SGDClassifierSGDClassifier*). Table 3.5 shows a list of the parameters used for training the model. When creating the documentation of a software library, we use this classifier to classify each sentence in the answers' text. Sentences labelled as positive are included in the documentation that is displayed to the user.

Table 3.6 shows the performance score of other classifiers built using different sets of features. The data set used in these experiments was a set of 250 manually labelled sentences, including 200 positive and 50 negative sentences.

Parameter	Value
penalty	"l2"
alpha	0.0001
l1 ratio	0.15
fit intercept	True
shuffle	True
epsilon	0.1
power t	0.5
learning rate	"optimal"

Table 3.5: Parameters used to build the SVM with SGD insight sentence classifier.

Classifier	Features	F-1 Score
Naive Bayes	Words, part-of-speech tags	0.31
Naive Bayes	Words, part-of-speech tags, score of the question, asker's reputation, score of the answer	0.33
Random Forest	Words, part-of-speech tags	0.36
Random Forest	Words, part-of-speech tags, score of the question, asker's reputation, score of the answer	0.39

Table 3.6: Classifiers built with different sets of features and their F-1 score.

3.5.2 Grammatical features of the sentence

In addition to satisfying the above definition of an insight sentence, in order for a sentence to be considered as an insight, it must be standalone, meaning it should make sense out of the context of the text where it came from. This characteristic is vital given that these sentences will be reviewed by developers who will see only the sentence, and not the context where it came from.

We can automatically identify whether a sentence is standalone or not using CoreNLP's co-reference annotator [11]. The co-reference annotator finds mentions of the same entity in a text, such as when "Theresa May" and "she" refer to the same person. The annotator implements both pronominal and nominal co-reference resolution.

We feed the paragraph containing the sentence to the parser and check whether the sentence has any dependencies to words outside its sentence. The following text snippet from Stack Overflow thread No. 1844688 helps demon-

Classifier	Best F-1 Score Obtained
Random Forest	0.678
Naive Bayes	0.684
SVM with SGD	0.714

Table 3.7: Insight sentences classifier results

strate this process.

> *"The example uses try-with-resources pattern recommended in API guide. It ensures that no matter circumstances the stream will be closed."*

Assume that we have identified the second sentence to be an insight sentence, using the approach that is explained earlier in this section. Through the results of the co-reference annotator, we can see that "*it*" at the beginning of the second sentence is referencing "*The example*". We remove the sentence from our set of results since it refers to a word from another sentence.

3.6 Results Aggregation

At this point, we aggregate the three components of our documentation in the form of a web page. Figure 1.3 shows the documentation for a given development task from library *junit*. We include three code snippets with the highest scores in this page, but more from the list of extracted code snippets can be added.

Chapter 4

Developer Survey

We conduct a developer survey to answer the following research questions:

- **RQ 1.** What is the quality of the information extracted by our methodology?
- **RQ 2.** How useful are the three components of our proposed documentation?
- **RQ 3.** What is the overall usefulness of a task-oriented library documentation?

To answer our research questions, which evaluate the quality of information and usefulness of our approach, we conduct a survey of Java developers. In this section, we describe the details of the survey setup.

4.1 Library Selection

In order to create task-oriented documentation, we had to select a number of libraries to use. We selected 10 Java software libraries, pertaining to five major domains of software development to use in the evaluation. We selected libraries from different domains to have a diverse dataset of questions and answers from Stack Overflow. For each domain, we selected well-known libraries that have a higher number of related questions on Stack Overflow. A list of these libraries and their respective domains is shown in Table 4.1.

Domain	Library Names
Testing	Testng Junit
Databases	Realm H2
Logging	Logback Log4j
Cryptography	Bouncy Castle Apache Shiro
XML Processing	JAXB Apache POI

Table 4.1: Libraries used for our survey.

4.2 Recruitment Strategies

To ensure a diverse set of participants for our survey, we used two recruitment strategies.

1. The first was sending an invitation email to graduate and undergraduate students of the Computer Science and Computer Engineering departments at our institution. This recruitment strategy ensures that we get participants who are more novice in terms of programming experience, and perhaps also familiarity with the target libraries.

2. The second recruitment strategy was targeted at more experienced developers who have used the libraries included in our survey. Such developers can better evaluate the quality of the information presented since they have used the libraries before. Additionally, having such a balance helps us understand if our task-oriented documentation is only useful for novice programmers, or if more experienced programmers also find it useful.

   For the second recruitment strategy, we searched for developers of GitHub who have either contributed to the repository of one of the software libraries in Table 4.1, or had used one of those libraries in one of their repositories. To find the list of contributors to libraries in our list, we used the list of contributors from the library's repository on GitHub.

To find people who had used the library, we used GitHub's search API [37] to find import statements that included the namespace of one of the libraries in Table 4.1, e.g. `import org.h2.tools.Server;`. We emailed 745 developers in total, asking them to participate in our survey.

4.3 Survey Setup

4.3.1 Overview of Survey Flow

We built a custom web application in order to have control over the flow of the survey and collecting responses of participants. We made sure that the participants understand and agree with our policy of using the data they provide in our work. Figure 4.1 shows the page where users consented to our privacy policy. We structure our survey as follows. In the first stage, all participants, regardless of how they were recruited, answer background questions about their occupation and Java experience. A snapshot of this stage of the survey can be seen in Figure 4.2.

At this point in the survey, one of two scenarios will happen:

1. If a participant was recruited using our first recruitment strategy, we select a library for them to review and ask them how familiar they are with this library. For each participant, we select the library with the fewest responses recorded so far. This selection process balances the number of responses received for each library, in order to avoid a large variance in the number of responses.

2. If a developer was recruited using our second recruitment strategy, we pre-select the library that was used to recruit them. They will evaluate tasks from this library, unless they choose to evaluate additional libraries before they exit the survey.

After the library selection process is completed, users had to indicate how much experience they have with the selected library, as can be seen in Figure 4.3.

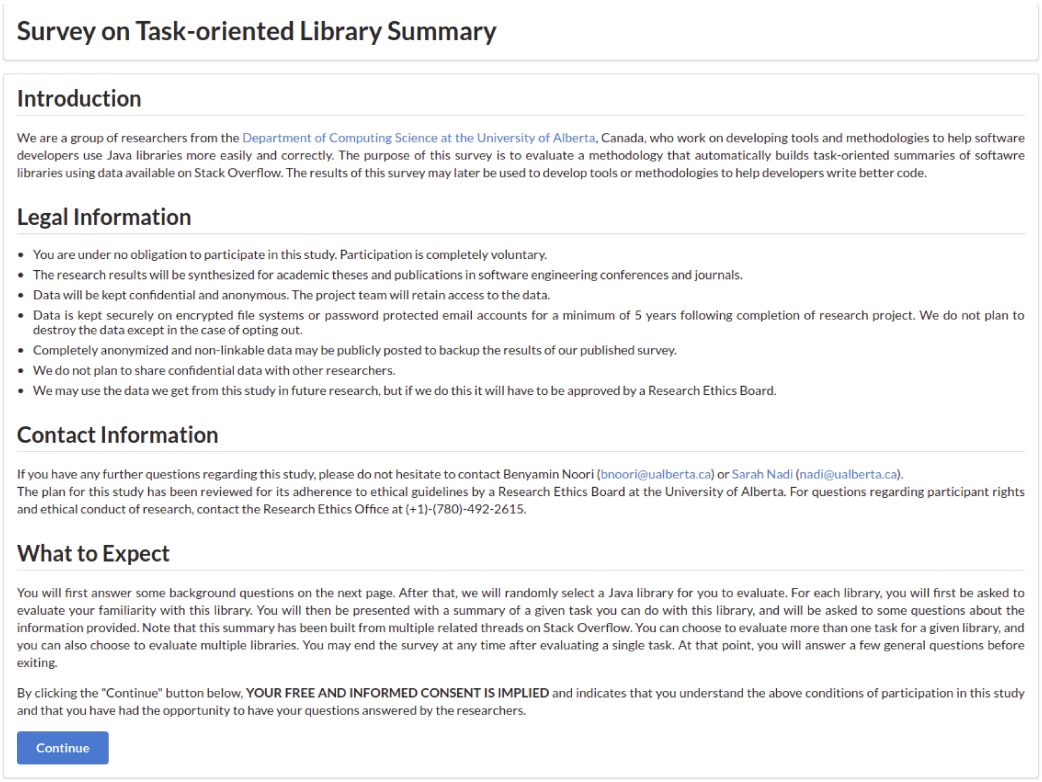

Figure 4.1: Our survey's privacy policy.

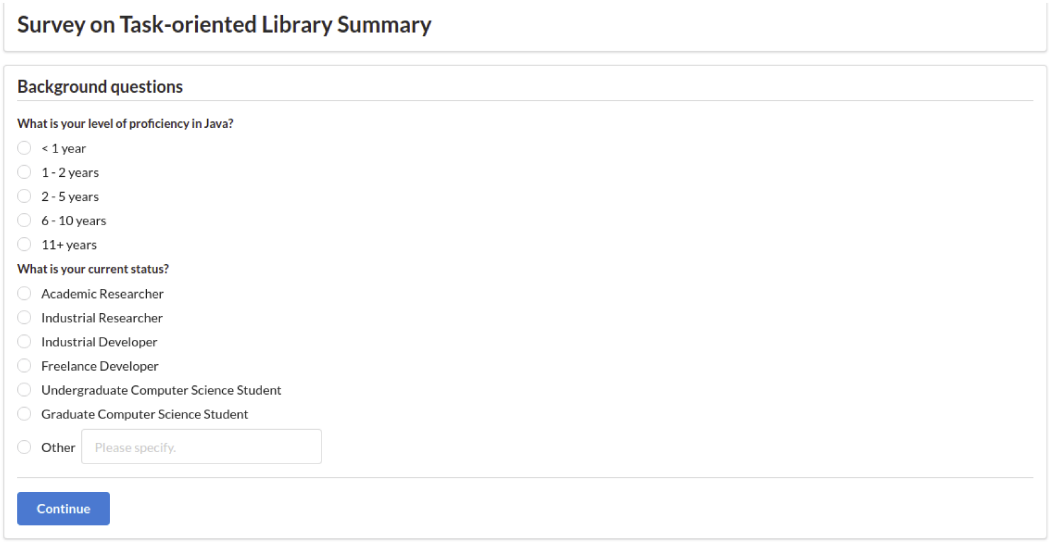

Figure 4.2: Background questions in our online survey.

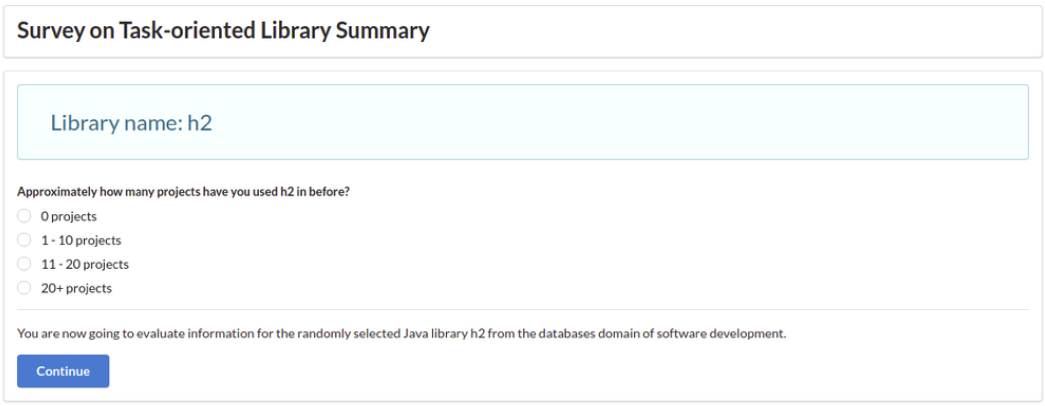

Figure 4.3: Participants indicated the number of projects in which they used their chosen library.

After answering the background questions, each participant proceeded to the second stage of the survey. In this stage, developers review documentation of software development tasks of the currently selected library. For each library, we selected the top three development tasks with the highest score on Stack Overflow to be displayed to the participants of our survey. For every task, we also display the three code snippets that belong to the three answers with the highest scores. For each software development task, developers are asked to review the task description, a set of code snippets as well as a number of insight sentences as can be seen in Figure 4.4.

The questions in this stage were designed to evaluate the quality of the information produced by our methodology. After reviewing each task, participants had two options: (1) Proceed to review another task. Each participant is able to review at most three tasks, or (2) Not review other tasks and immediately move on to stage three, final questions.

In the final stage of the online survey, we asked developers final questions about the usefulness of the documentation and its components. Figure 4.5 provides a view of the final page of our survey.

These questions were designed to evaluate the usefulness of task-oriented documentation overall as well as by component. We also gave participants the option of providing us with open ended text comments. After answering the final questions, participants could go back to review the documentation of

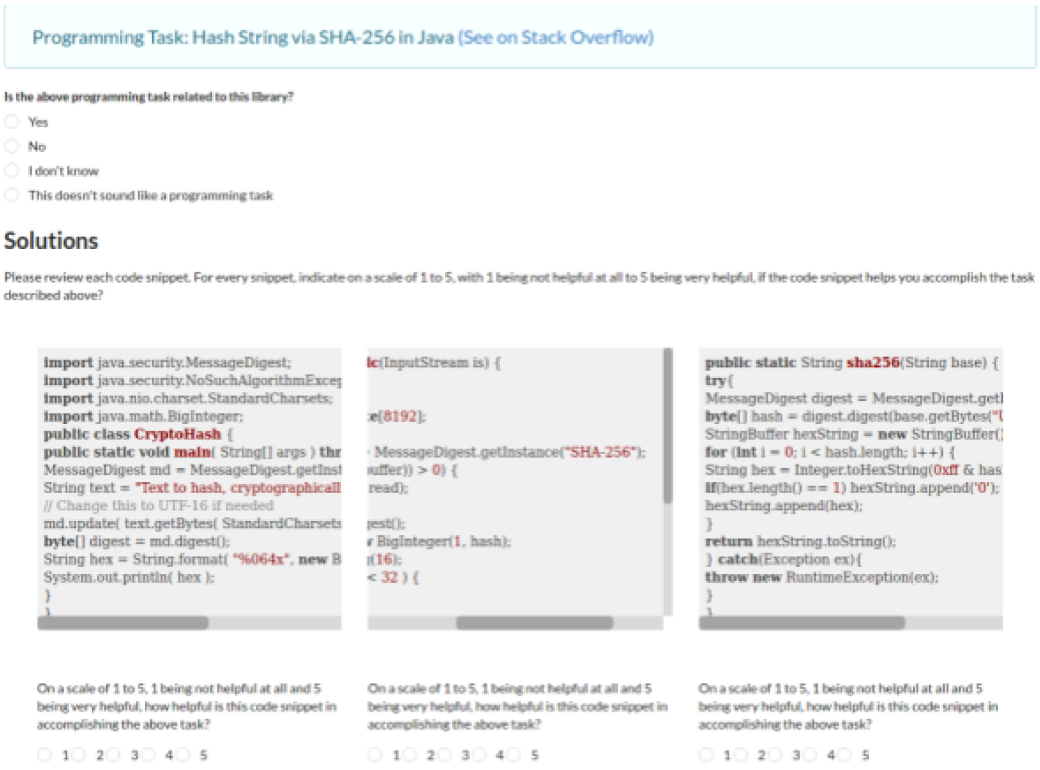

Figure 4.4: Participants reviewed task descriptions, code snippets and insight sentences.

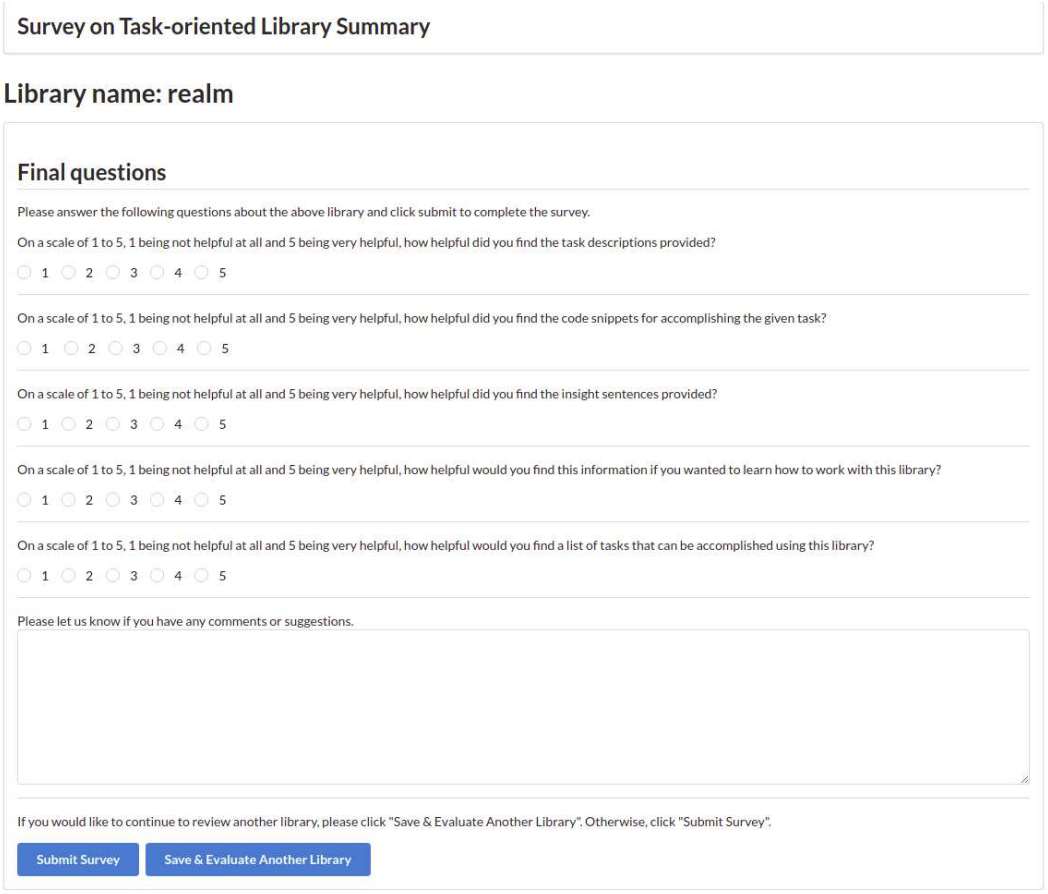

Figure 4.5: Final questions were intended to evaluate the users' overall perceived usefulness of the components of the documentation.

another library or exit the survey.

4.3.2 Details of Survey Questions

The following is a summary of our survey questions:

1. Stage 1: background questions
   (a) What is your current occupation? *Undergraduate Student, Graduate Student, Academic Researcher, Industrial Researcher, Industrial Developer, Freelance Developer.*
   (b) How many years of Java development experience do you have? *Zero, Less than 1, 1 to 2, 2 to 5, 6 to 10, 11+.*
   (c) Approximately how many projects have you used $<$ *library* $>$ in before? *Zero, 1 to 10, 11 to 20, 20+.*
2. Stage 2: development task reviews
   (a) Is the programming task related to this library? *Yes, No, I don't know, This doesn't seem like a programming task.*
   (b) How helpful is this code snippet in accomplishing the task?
   (c) How helpful is this sentence when using this library?
3. Stage 3: final questions:
   (a) How helpful did you find the task descriptions provided?
   (b) How helpful did you find the code snippets for accomplishing the given task?
   (c) How helpful did you find the insight sentences provided?
   (d) How helpful would you find this information if you wanted to learn how to work with this library?
   (e) How helpful would you find a list of tasks that can be accomplished using this library?

(f) Please let us know if you have any comments or suggestions. *free-text*

Answers to questions 2.b, 2.c, 3.a, 3.b, 3.c, 3.d, and 3.e were provided as a score on a Likert scale from 1 to 5, with 1 being not helpful at all and 5 being very helpful.

Chapter 5

Survey Results

We received a total of 69 responses, 31 responses from our first recruitment strategy (unknown response rate because of sampling strategy) and 38 from the second (˜5% response rate). Note that we received many more responses that only completed the background questions so our response rate is technically higher. However, since these participants did not evaluate any tasks, we do not include them in any of our calculations.

The 69 responses we obtained include reviews for 125 tasks. Note that this means that some participants chose to exit the survey without evaluating all three tasks. Most participants (˜56%) only reviewed one development task. Collectively, participants reviewed 306 code snippets and 262 insights sentences.

Figures 5.1, 5.2 and 5.3 present the distribution of the background of our survey participants. Graduate computer science students and industrial developers represent the two major groups of participants, as shown in figure 5.1.

Figures 5.4 shows the number of responses received per library. As explained previously, in our first recruitment strategy, we select libraries to be reviewed using a process that aims to balance the number of responses per library. However, in our second recruitment strategy, the number of invitations per library depends on external factors, such as the number of contributors in a library's repository. That is one possible reason for the uneven distribution of number of responses per library. We now use the results of the survey to

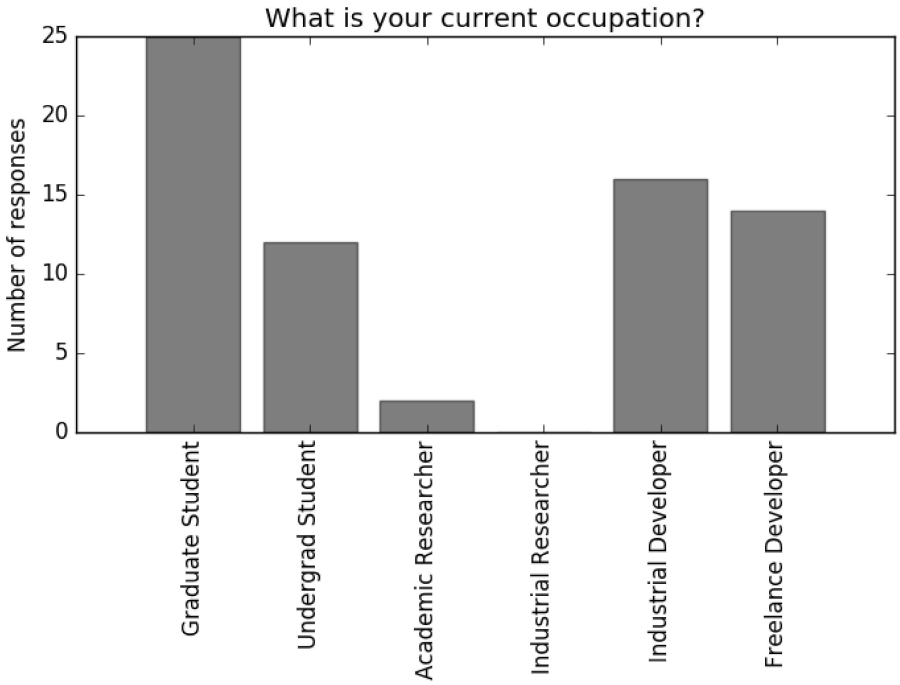

Figure 5.1: Occupation of participants.

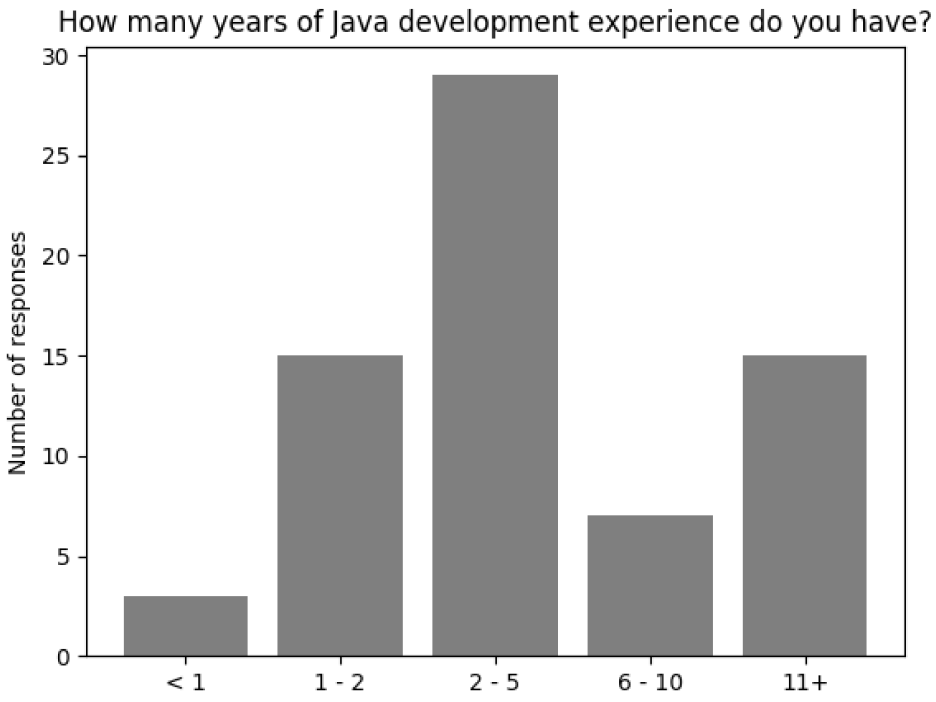

Figure 5.2: Participants' experience with Java development.

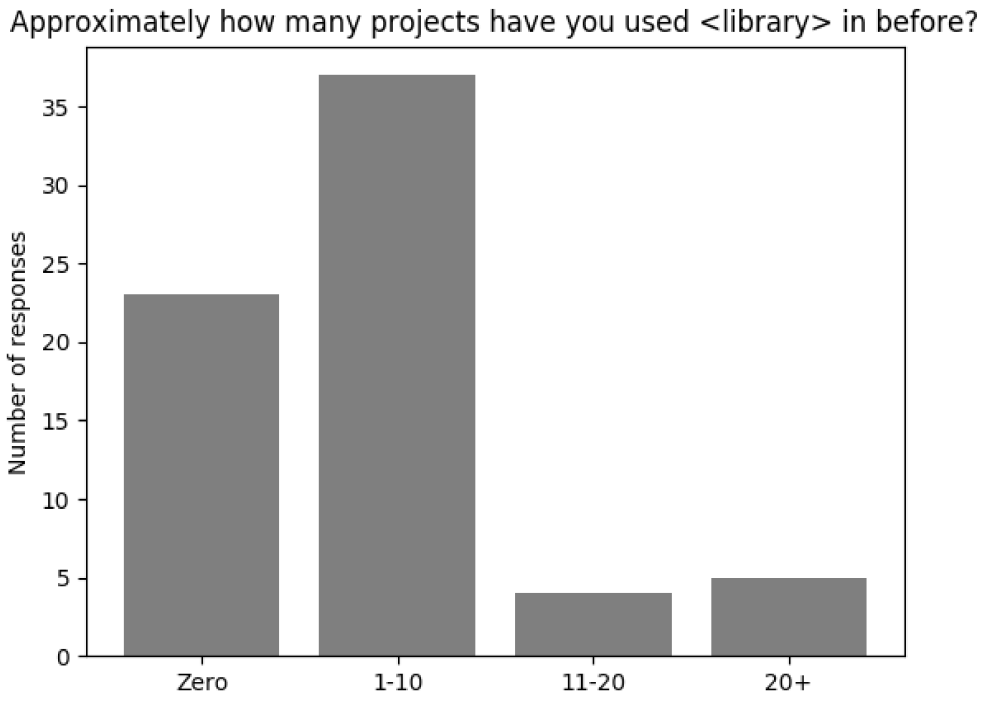

Figure 5.3: Number of projects with the selected library.

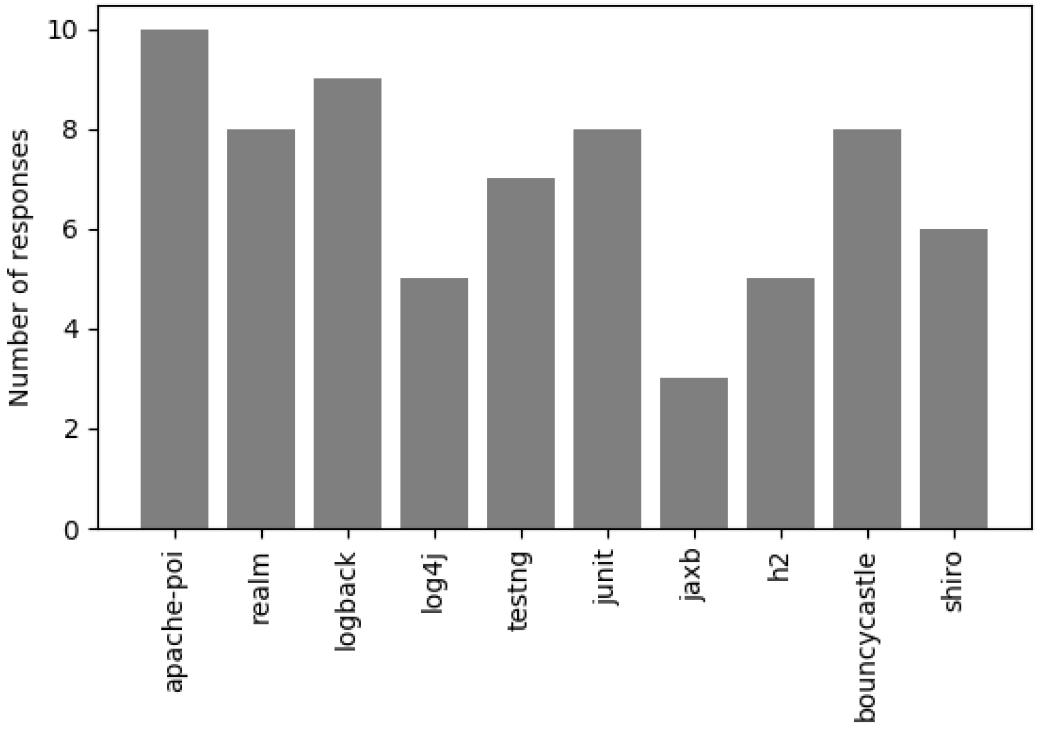

Figure 5.4: Number of responses received per library.

answer the three research questions presented in the introduction.

5.1 Quality of Information

In this section, we use the responses of our survey to evaluate the quality of information extracted using our methodology. We do this by analyzing the answers to questions answered in the second stage of the online survey.

Our definition of high-quality task-oriented documentation is one that provides correct information that is related to the task and library at hand, and that is helpful in accomplishing the task. Asking participants to evaluate correctness is difficult, since many of them may not have used the given library for the given task before. We, therefore, focus on evaluating how related a given task is to the specified library, and how helpful the code snippets and insight sentences are with respect to the given task. Therefore, to answer RQ1, we look at the three components of our documentation separately.

5.1.1 Task Description

Figure 5.5 shows the summary of participant responses for evaluating whether a given task is related to the library in question. Note that this figure includes all responses received including multiple responses for the same task description. Therefore, if three participants responded "Yes", "Yes", and "No", these responses were counted as 2 in the "Yes" column and one in the "No" column.

The majority of reviewed tasks were found to be related to the respective library. Specifically, participants found that 73.6% of the 125 reviewed tasks were related to the library in question. A few responses (1.6%) said the description provided does not seem like a software development task. Thirteen (10.4%) responses said the task was not relevant to the library, and 18 responses (14.4%) indicated that they did not know whether the task was relevant or not.

Our results suggest that the task extraction methodology we use works desirably a majority of the time on the data we extracted from Stack Overflow. However, since some of the tasks were marked as not related or that they do not seem to be a task, there is still room for improvement. We further discuss this in chapter 6.

5.1.2 Code Snippets

To answer RQ1 with respect to the second component of our task-oriented documentation, we look at the 306 code snippets reviewed by our survey participants.

When reviewing the presented tasks, participants indicated the helpfulness of code snippets in accomplishing the task described, provided on a scale of 1 to 5. Figure 5.6 shows the distribution of ratings for all code snippets, divided by the corresponding library. On average, our extracted code snippets had a rating of 3.49 out of 5. The median of the ratings was 4, suggesting that participants typically perceived the provided code snippets as useful. The interquartile range of the responses received was 2.

We performed a Chi-Square test of independency to see if the distribution

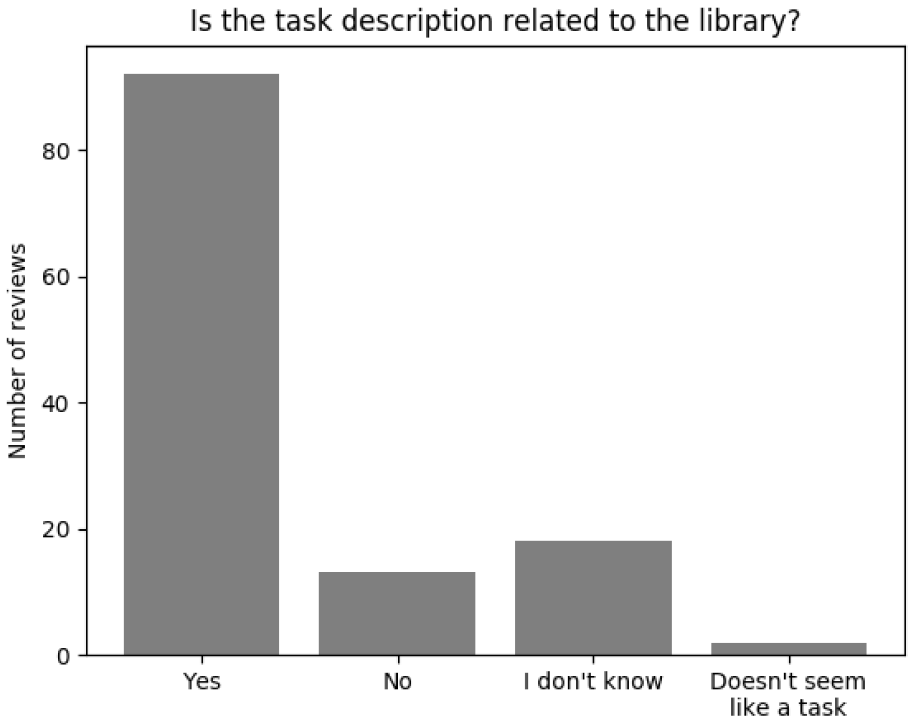

Figure 5.5: Task description reviews.

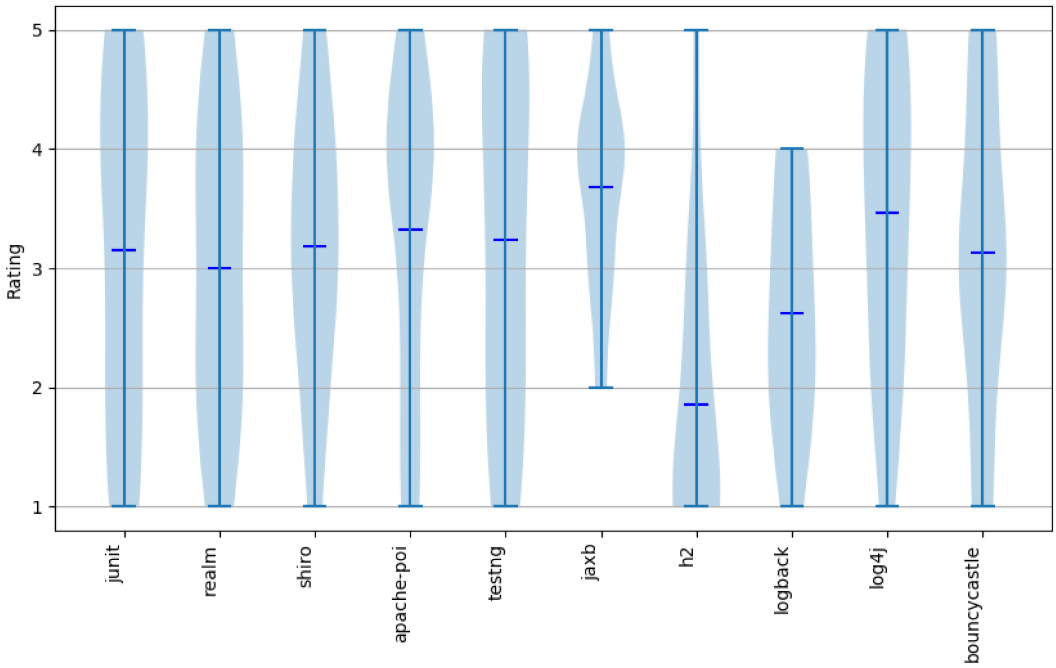

Figure 5.6: Code snippet reviews per library.

of ratings was significantly different between libraries. We ran this test for the ratings of every pair of libraries and the lowest p-value obtained for a pair of libraries was 0.3207, which is not statistically significant, meaning it does not show any such correlation exists.

5.1.3 Insight Sentences

Insight sentences received an average score of 3.25 out of 5 for usefulness, making them the least useful component in the generated documentation. The median of insight sentence reviews was 3 and the interquartile range of the responses was 2.

Two participants left comments saying insight sentences only had value when viewed in their original context on Stack Overflow. In our definitions of insight sentences, we specify that an insight sentence must be standalone, meaning it should make sense outside of its context. The results of the survey show that while our current extraction methodology identifies useful sentences, the sentences do not necessarily satisfy the standalone criteria. We can conclude that insight sentences as a component are potentially useful but they need to be further improved to make sense to the developer outside of their contexts.

We performed a Chi-Square test of independency to see if the distribution of ratings was significantly different between libraries. We ran this test for the ratings of every pair of libraries and the lowest p-value obtained for a pair of libraries was 0.2111, which is not statistically significant, meaning it does not show any such correlation exists.

5.2 Usefulness of Information

We now answer RQ2 and RQ3 by looking at three sources of information.

First, we use participants' answers to the question in the exit questionnaire about how helpful would they find the provided summaries for the purpose of learning to work with a library. We find that the mean rating for this answer is 3.17 out of 5. The responses to this questions had an interquartile range of

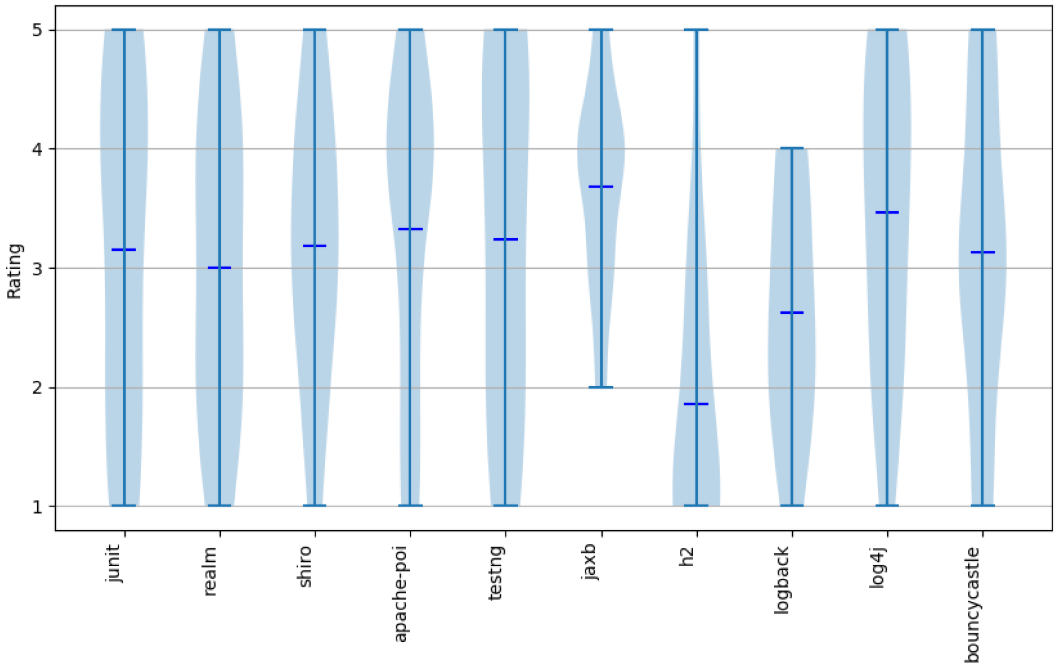

Figure 5.7: Insight sentences reviews per library.

2 and a median of 3.

Second, we use participants' answers to the question in the exit questionnaire about how useful they would find a list of tasks each library can accomplish. While we designed the survey in a way that participants can evaluate one task at a time, our idea of the actual documentation would be to list all tasks related to a given library, as in Figure 1.2, and when the developer clicks on a task, they would see the information we show in Figure 1.3. However, for surveying purposes, leaving participants to randomly select one of these tasks to evaluate is not feasible in terms of balancing responses. By asking this second question to participants, we can evaluate whether they would find a task-oriented summary of the library useful. This was indeed the case, where the average score received for this question was 3.67 out of 5 with a median 4 of and interquartile range of 1. Forty-four (63.76%) responses gave a score of 4 or 5.

Finally, we also looked at the free-text survey comments some participants left. One participant elaborates on how a task list would be useful to them *"To*

avoid spending time on the whole documentation, finding a list of tasks (from basic to advanced) that can be accomplished using a library is the first thing that I would be looking for when I need to work with the library." Another participant also says that they would find this task list useful, if it is more generalized to contain information about which libraries accomplish this task. The ranking of the task list and the comments of these participants confirm our intuition that have a form of task-based documentation would be helpful to developers.

5.2.1 Relationship Between Backgrounds and Responses

We also investigated any statistical correlation between the participants' backgrounds and how they answered the questions of our survey. We ran a chi-squared correlation test on the results of the survey, and the test showed no correlation between any items of the participants backgrounds and the answers provided.

Chapter 6

Implications

In this chapter, we discuss possible implications of our results as well as our findings. We use our findings to present improvements to parts of our methodology. Based on the results of the survey, we can conclude that developers perceive task-oriented documentation of software libraries as useful. This is also backed up by comments received from the survey indicating that a list of tasks would be an important and useful component.

Software developers rated the code snippets extracted from Stack Overflow as the most useful component of the task-oriented documentation. The next most useful components were task descriptions. Even though our study shows that task-oriented documentation is a useful concept for developers, our results also suggest that there is still room for improvement. In this chapter, we further analyze our results and discuss the implications that can be inferred.

6.1 Task Description

Even though a majority of the participants of our study found task descriptions useful, a few of the responses indicate it did not always perform desirably. We examine cases where participants provided a negative rating for the task descriptions and look at the comments they wrote. Some participants struggled with task descriptions and had to refer to the original thread on Stack Overflow to realize the details of what was being discussed. Others said the task description was either not relevant or not coherent (marked *"doesn't seem like a task"*). The following are a few possible causes of this phenomenon:

1. The question titles of Stack Overflow threads are not always grammatically correct. Given that our task extraction methodology heavily depends on CoreNLP's outputs, this can impact the process negatively. This was in fact the cause a number of low ratings we investigated. For instance, a title from a Stack Overflow thread was *"persistence.xml to import database parameters values from .properties file"*. While this sentence is describing a development task, it is not grammatically coherent and the resulting task description is not ideal, leading 4 participants to indicate they do not think this is a development task.

   One approach to improve in the future is to use NLP tools that can detect and fix grammatical mistakes [9], [10], [13].

2. In some cases, the task description does not have enough information for a user to understand the task at hand. Users on Stack Overflow compensate for this lack of information by referring to the body of the question, where the poster usually adds details and context to specify the development task. We can address this by extending our methodology to include key pieces of information from the question body into the task description.

   While this will help narrow down the development tasks and will make this component more useful than its current state, it is not a trivial task. There are general text summarization techniques that can used [2], [15], [52], but adjusting them for our purposes can be challenging. For example, it has been shown that Lex-rank [15], a stochastic graph-based method for computing relative importance of textual units, does not perform desirably when applied to text collected from Stack Overflow [46].

6.2 Code Snippets

Code snippets were the most useful component of our proposed documentation. However, Figure 5.6 shows that not all libraries had the same standard deviation of scores. While the figure shows that the mean rating of code snip-

Library	Mean	Standard Deviation
JUnit	3.45	1.45
Realm	3.11	1.51
Shiro	3.55	1.12
Apache-POI	3.75	1.03
Testng	3.5	1.51
JAXB	3.70	0.83
h2	3.65	1.39
Logback	3.54	1.05
Log4j	3.0	1.39
Bouncycastle	3.66	1.10

Table 6.1: Mean and standard deviation of ratings of code snippets per library.

pets for all libraries is between 3 and 4, we can see that some libraries have lower means that others. The standard deviation of ratings of code snippets for every library can be seen in Table 6.1.

As we can see from the table, *log4j* seems controversial, with ratings varying from 1 to 5 on the scale. On the other hand, ratings for *JAXB* and *bouncycastle* are more consistent with lower standard deviation. We further investigate the tasks and responses for *log4j* to understand these differences. We find that one of the threads with the highest score for *log4j* was tagged only with *"log4j"*, because the user who asked the question used that library for logging, while the main purpose of the thread was to convert milliseconds to time format. Since we rely on the tags on Stack Overflow, such a thread would be included in our documentation. However, since it is not related to *log4j* and the code snippets do not contain anything related to this library, it received lower scores from users who reviewed the documentation of this particular thread.

Code snippets receiving a low rating score may have also been extracted from less popular answers on Stack Overflow. Even though we sort answers based on their scores, a thread may only contain only a few code snippets, which would allow snippets from answers with low scores to be included in a task documentation. This was in fact the case in some of the reviews that we examined. Comments by participants also suggested that sometimes the snippets are of less than desirable quality or do not use the specified library.

These problems can be addressed by accounting for the specific classes or

Library	Mean	Standard Deviation
JUnit	3.15	1.41
Realm	3.0	1.43
Shiro	3.18	1.21
Apache-POI	3.32	0.98
Testng	3.24	1.40
JAXB	3.67	0.86
h2	1.85	1.23
Logback	2.62	1.16
Log4j	3.46	1.26
Bouncycastle	3.13	1.08

Table 6.2: Mean and standard deviation of ratings of insight sentences per library.

packages used in code snippets. This idea has been explored in Baker [42], and our methodology can be extended to incorporate it. We basically need to find code snippets on Stack Overflow that use one of the APIs of our target library. However, based on initial experimentation, we found that `Baker` does not always provide perfect precision, and a code snippet might end up having more than one related API. It is still not clear what is the best heuristic to use to select the "main" API to tag this snippet with; otherwise, the majority of code snippets would be tagged with APIs from the standard Java library. Future work can also investigate additional code snippet quality metrics other than the answer score.

6.3 Insight Sentences

Insights were deemed useful as a component by the participants of our survey. They also had the lowest average score in task documentation reviews (stage two of the survey) by participants. This indicates that while software developers find insights from Stack Overflow useful as a component, our methodology has room for improvement.

While the mean score across all insight sentences indicates a good quality, Figure 5.7 shows that the average rating per library varies between different libraries.

In Figure 5.7, Library *h2* has the lowest mean with very few sentences receiving a score of 4 or 5, while *jaxb* has the highest mean with very few sentences receiving a score of 1 or 2. One possible reason for this is that insights used to build the documentation of *jaxb* came from answers with higher scores compared to the answers used for *h2*. Since we present a summarization approach of existing data, our intuition was to rank information based on score rather than filter them out if their score falls below a certain threshold, and the user would see the high ranked information first after all and that could affect the way the users rate the insight sentences.

This is because some libraries are more popular than others and the scores vary significantly between libraries, which makes it harder to choose a reasonable threshold metric. One possible way to solve this is to calculate a normalized score for each answer, depending on the highest score any answer for this library received. We also further investigated the documentation summaries and the ratings to determine other causes of this difference between the scores for insight sentences. In some of the cases where the participant provided a low score for an insight, the possible reason was one of the following:

1. The insight sentence had a reference to an element outside of itself, i.e. it was not standalone. An example of such a sentence is *"Using junit, I was able to write this simple test."*. This sentence is referring to a code snippet following it. This insight received a very low average score of 1.0.

2. The sentence had grammatical errors. This was observed in a number of cases. For instance the following insight received an average score of 1.2 out of 5: *"If you don't stop it, Web Application Server never ending until you kill it."*. Even though this sentence has useful information embedded (until you explicitly terminate the web server process, it will remain running), it is possible that the sentence was rated negatively due to its grammatical incoherence. We presented sentences as they appear on Stack Overflow, and did not try to detect grammar errors. This can be an interesting quality characteristic for further narrowing down good

insight sentences.

3. The sentence was misclassified by the machine learning model. The F-1 score of our best model was 0.714 and that means sentences can be misclassified as an insight when they are not in fact insight sentences. As mentioned before, we did explore using additional features to improve the performance of our classifier, but these features did not have a positive effect.

   Based on the results of the survey, we suggest potential improvement ideas beyond what we explored. One such possibility, for instance, is to use a word to vector representation model that represents words in the form of vectors [23]. Such vectors have been shown to bear information regarding the context of words in the sentence and can be helpful for our learning model [24], [25]. This idea has also been explored in the context of software engineering [7]. However, applying this vector representation in a way that benefits our methodology is a non-trivial task.

Chapter 7

Threats to Validity

In this chapter, we discuss the threats to the validity of the survey results we presented in Chapter 5.

7.1 Internal Validity

Internal validity is the extent to which a conclusion based on a study is warranted, which is determined by the degree to which a study minimizes bias [3].

A possible threat to the shown information is the presence of errors or bugs in our scripts used to extract information from the raw data collected from Stack Overflow. To mitigate this threat, we manually verified various samples of the results. We also share all our scripts and data online for replication and verification purposes.

To create a training set of insight sentences, we manually classified a sample of sentences collected from Stack Overflow. This could have an impact on the predictions of the classifier. To mitigate this threat, two authors separately tagged the sentences while having our criteria list in mind, and then discussed any disagreements.

In order to find all threads on Stack Overflow related to a library, we assume that all threads associated with that library's specific tag are related to it. While that is not the case at all times, as discussed in Chapter 6, it is often the case that the threads with the highest score often discuss the more widely used and important development tasks accomplished with that library. In fact, in the 10 libraries we used in our survey, there were only a few cases

of a thread not being related to its designated library.

As developers proceed through the survey, they review task descriptions, code snippets and insight sentences and become more familiar with the system and that could affect the way they respond to our questions. Also, given our recruitment strategies, some participants are not familiar with the library that was selected for them to review, which could be another factor that impacts their responses.

7.2 External Validity

A threat to external validity is an explanation of how you might be wrong in making a generalization [48].

We applied our task-oriented documentation methodology to ten popular libraries from five major software development domains. However, our methodology heavily relies on collection of data from Stack Overflow. As discussed in chapter 6, the quality of information collected from Stack Overflow can have an impact on the findings of our work. We know that for less popular or less well-known libraries, there will be fewer threads posted on Stack Overflow. This leads to fewer development tasks and code snippets, which leads to a less comprehensive documentation of the library. However, by adjusting some of our extraction techniques, we can extend our methodology and apply it to other sources of information such as library documentation and tutorials.

Our findings and conclusions presented in this work are based on the results of surveying 69 Java developers based on only 10 libraries. To reduce possible opinion biases, we recruited participants with varying backgrounds and through different sources. Our survey was based on data extracted for only 10 Java libraries. While we cannot generalize our results beyond these libraries, our library selection was diverse, covering various domains of software development.

7.3 Construct Validity

Construct validity is the degree to which a test measures what it claims, or purports, to be measuring [12].

Given that in our developer survey, we do not ask participants to use the generated task-oriented library documentation in everyday tasks and compare the results with conventional documentation or other sources of information about libraries, we measure only perceive usefulness of the proposed methodology and results.

The results produced in this work show that software developers of different backgrounds find the concept of task-oriented documentation useful but do not draw comparison between task-oriented documentation and other types of documentation.

Note that some of the terms used in the online survey can be subjective and different participants might have different ideas of what they mean, specially in the context of software library documentation. The most important examples of such terms are “usefulness” and “helpfulness”. This can impact the scores developers give in our survey.

Chapter 8

Improvements to Insight Sentence Extraction

As stated before, our survey of software developers revealed that insight sentences as a component were perceived as useful, but the insight sentences extracted using our methodology did not receive desirable scores for perceived usefulness. In this chapter, we describe an additional part of our work whose focus is solely on extracting valuable information from text in Stack Overflow answers. This methodology is designed to help extract more focused insight sentences of higher quality.

8.1 Introduction

In order to approach the problem, we narrowed down our definition of insight sentences. We studied a large sample of randomly sampled text from Stack Overflow answers and identified different types of insight sentences:

1. *Example*: These sentences provide an example of how something would work or what the output would be, sometimes referring to a methodology or idea explained prior to the example.

   An example sentence usually contains phrases like: "for example", "for instance", and often contains references to an outside element, sentence, code snippet. The following is an instance of example insight sentences:

   For example, parsing "1600 Amphitheatre Parkway, Mountain

View, CA" yields: [followed by code snippet] [31]

2. *Suggestion*: These sentences suggest a library, method or class to accomplish the task at hand. They describe either the advantages or disadvantages of a given solution. The following sentence is an example of this type of insight:

   > If you want to download the data at background, you can use the BackgroundDownloader class [54].

3. *External*: Such sentences link to external resources (i.e. outside of the current Stack Overflow thread) that need to be studied. This sentence is an example of this type:

   > Look at the example from `http://en.wikipedia.org/wiki/Algebraic_data_types` [16].

4. *Conditional*: Sentences in this group recommend a certain strategy based on some condition. Such sentences usually contain "if" statements. The following example from Stack Overflow shows is one such sentence:

   > If you want to see which styles are affecting a particular element, the Web Developer Toolbar for firefox has a handy Style Information command for seeing which styles (from which files/style blocks/inline styles) are being applied to it [18].

5. *Details*: Discuss nuanced details of a technology, like the following sentence:

   > In C#, #define macros, like some of Bernard's examples, are not allowed [5].

Of the classes mentioned above, the conditional class is the easiest to detect automatically, given its grammatical features. In our work in this chapter, we focus on conditional insight sentences. Our methodology aims to extract these

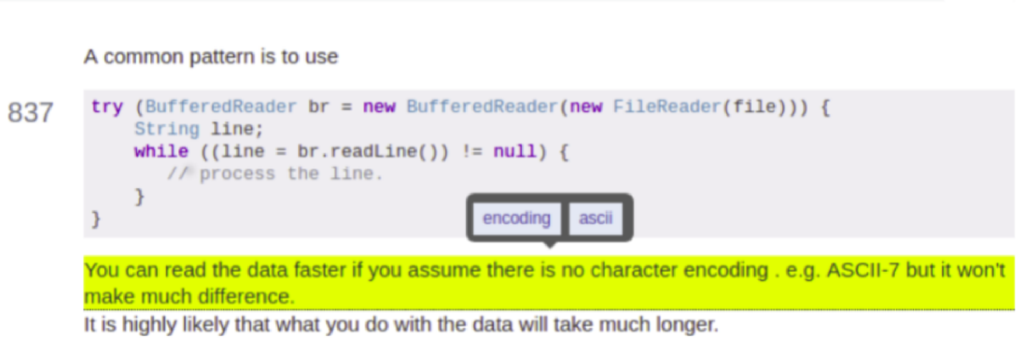

Figure 8.1: An example of a system utilizing conditional insights extraction to help developers navigate Stack Overflow.

sentences and process the information in a way that can be of use to a software developer using Stack Overflow to find information necessary to accomplish their daily development tasks.

Our vision is that identifying such sentences and associating them additional information such as technical words (e.g. programming language, framework, library, etc.) or non-functional aspects can be useful for navigating Stack Overflow information and can be used in the future to narrow down search results.

For instance, if a user on Stack Overflow wants to find answers discussing the encoding of a given file, we can use information associated with conditional insight sentences to help them find answers relevant to the topics they're interested in. The same can be said about non-functional aspects such as performance and security. Figure 8.1 shows an example of a system that uses highlighting of sentences on Stack Overflow to help developers find the necessary information with more ease and speed.

8.2 Extracting Conditional Insight Sentences

As mentioned before, users on Stack Overflow describe their questions with a set of tags. First, we acquire a data storage containing all the tags on Stack Overflow. We obtained the most recent data dumps of all of Stack Overflow's data available [38]. This data storage is later used in processing of

conditional insight sentences. We use this because tags on Stack Overflow are often associated with various types of technologies, programming languages, libraries, concepts, etc.

We also use a list of keywords describing non-functional requirements. This list of keywords was created by Hindle et al. [17] to automatically categorize software maintenance activities into different categories. This list can help us categorize conditional sentences based on any non-functional aspects they may discuss.

Given a Stack Overflow thread, our goal is to identify all conditional insight sentences among the sentences in the answers of that thread. In order to identify conditional insight sentences, we make use of CoreNLP's constituency parser [30]. This functionality provides full syntactic analysis, minimally a constituency (phrase-structure tree) parse of sentences. Using outputs provided by this parser, we can identify whether a sentence contains a conditional structure. If so, we can also identify the conditional clause as well as the main clause.

The following are the steps in our conditional insight sentence extraction process:

1. We begin the processing of each Stack Overflow thread by extracting all the text from all of its answers. We then feed all the text to Core NLP's constituency parser. When we encounter a conditional sentence, we further process that sentence to extract the desired information.

2. The conditional part of the sentence contains a description of the condition. In this stage, we obtain the part-of-speech tags for words in the conditional part, and store all the nouns separately.

3. If any of the aforementioned nouns are words that are also Stack Overflow tags, we associate the conditional sentence at hand with that tag. We can do this simply by searching for the nouns in the tags described earlier in this section.

4. We also match the words in the set of nouns to words in the list of non-

functional requirements. If any matches are found, the non-functional requirements are associated with the conditional sentence at hand.

5.

As stated before, the purpose of associating these sentences with tags and non-functional requirements is that it allows future methodologies to categorize these sentences based on these associations. For example, the associated non functional requirements can be used to narrow down search results based on specific criteria provided by the user.

8.3 Evaluation

In order to evaluate the precision of our improvements to the insights extraction, we prepared a simple evaluation to assess the precision of the extracted sentences. We processed text collected from answers of 100 randomly sampled threads. We then processed those sentences using our improved insight sentence extractor. This gave us 451 extracted insight sentences.

Then, two human annotators labelled each of these sentences as either a conditional insight sentence (positive class) or not (negative class). When labelling these sentences we used the following definition:

> A conditional insight sentence, is a conditional sentence that if not read by the user, will lead to a less than ideal solution.

There were 60 conflicts in total in labelling these sentences. In those cases, the annotators discussed the situation and came up with a final resolution as the label of that sentence. After the labelling was complete, we found 323 true positive and 128 false positive sentences. The Kappa agreement score in this labelling was 0.67.

Precision is the percentage of true positive sentences, i.e. those correctly labelled positive by the extraction process. Based on our manual validation, the precision of our improved insight sentence extraction was 71.61%. While

the precision obtained in this evaluation is promising, a manual review of the results can point to what needs to be done in order to improve the methodology.

For example, a part of the false positive examples were sentences that had were questions (in the text of Stack Overflow answers) that had the word "if" in their structure. Often these questions provided little to no insight and were therefore labelled negative by annotators. "If you look at the adblock site there is some indication of how it does blocking: How does element hiding work?" is one such sentence.

Chapter 9

Conclusions & Future Work

There exists an information gap between the information needed by software developers and what is provided by the documentation. We introduced a methodology to automatically leverage information on Stack Overflow to create task-oriented documentation of software libraries that can fill this gap. The proposed task-oriented library documentation has three main components: (1) a set of software development tasks the library can accomplish, (2) a set of code snippets per development task demonstrating different methods of accomplishing the task, and (3) insights that discuss details of different approaches.

We applied our methodology to 10 well known Java libraries from 5 software development domains and evaluated the results using a survey of 69 Java developers. Our results indicated that developers perceive our proposed task-oriented documentation as useful. However, our results also shed light on current limitations and challenges of processing crowd-sourced information. We provided concrete discussions of these challenges and suggestions for overcoming them to improve the extraction and generation of software documentation. Given the findings of our survey, we further explored insight sentences and described an improved methodology for identifying conditional insight sentences. Our improved methodology made use of grammatical features of sentences on Stack Overflow to identify a specific class of insight sentences, conditional insights.

The future work in this area can have multiple directions. One possibility is investigating whether task-oriented documentation is better than conventional

documentation for performing software development tasks. This investigation can include an experiment in which a group of software developers accomplish software development tasks using task-oriented documentation while another group perform the same tasks using conventional documentation.

Another direction is to build a system of navigation using the outputs of our proposed conditional insights extraction and investigate whether such a system helps users find the information they're looking for with more ease and speed. The suggestions for improvement provided in this thesis can also be implemented and evaluated in such a study.

References

[1] 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*, vol. 19, no. 3, pp. 619–654, 2014. 11

[2] F. Boudin, M. El-Bèze, and J.-M. Torres-Moreno, "A scalable mmr approach to sentence scoring for multi-document update summarization," *Coling 2008: Companion volume: Posters*, pp. 23–26, 2008. 46

[3] M. B. Brewer and W. D. Crano, "Research design and issues of validity," *Handbook of research methods in social and personality psychology*, pp. 3–16, 2000. 51

[4] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, no. 8-13, pp. 1157–1166, 1997. 19

[5] *C - how do you use define? - stack overflow*, `https://stackoverflow.com/questions/15744/how-do-you-use-define`, Accessed: 2018-08-1. 55

[6] B. A. Campbell and C. Treude, "Nlp2code: Code snippet content assist via natural language tasks," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, IEEE, 2017, pp. 628–632. 9, 11

[7] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in q&a discussions–incorporating relational and categorical knowledge into word embedding," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, IEEE, vol. 1, 2016, pp. 338–348. 50

[8] C. Chen and K. Zhang, "Who asked what: Integrating crowdsourced faqs into API documentation," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 456–459. 1, 10, 11

[9] M. Chodorow and C. Leacock, "An unsupervised method for detecting grammatical errors," in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, Association for Computational Linguistics, 2000, pp. 140–147. 46

[10] M. Chodorow, J. R. Tetreault, and N.-R. Han, "Detection of grammatical errors involving prepositions," in *Proceedings of the fourth ACL-SIGSEM workshop on prepositions*, Association for Computational Linguistics, 2007, pp. 25–30. 46

[11] *Corefannotator — stanford corenlp*, `https://stanfordnlp.github.io/CoreNLP/coref.html`, Accessed: 2018-06-18. 24

[12] L. J. Cronbach and P. E. Meehl, "Construct validity in psychological tests.," *Psychological bulletin*, vol. 52, no. 4, p. 281, 1955. 53

[13] R. De Felice and S. G. Pulman, "A classifier-based approach to preposition and determiner error correction in l2 english," in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, Association for Computational Linguistics, 2008, pp. 169–176. 46

[14] U. Dekel and J. D. Herbsleb, "Improving API documentation usability with knowledge pushing," in *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, IEEE, 2009, pp. 320–330. 1, 9

[15] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004. 46

[16] *Haskell's algebraic data types - stack overflow*, `https://stackoverflow.com/questions/16770/haskells-algebraic-data-types/35758`, Accessed: 2018-08-1. 55

[17] A. Hindle, N. Ernst, M. W. Godfrey, R. C. Holt, and J. Mylopoulos, "What's in a name? on the automated topic naming of software maintenance activities," *submission: http://softwareprocess. es/whats-in-a-name*, vol. 125, pp. 150–155, 2010. 57

[18] *Ie7 html/css margin-bottom bug - stack overflow*, `https://stackoverflow.com/questions/15326/ie7-html-css-margin-bottom-bug/19264`, Accessed: 2018-08-1. 55

[19] H. Jiang, L. Nie, Z. Sun, Z. Ren, W. Kong, T. Zhang, and X. Luo, "Rosf: Leveraging information retrieval and supervised learning for recommending code snippets," *IEEE Transactions on Services Computing*, 2016. 9

[20] *Junit 5*, `https://junit.org/`, Accessed: 2018-06-18. 12

[21] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE software*, vol. 20, no. 6, pp. 35–39, 2003. 1

[22] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60. 14

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. 50

[24] ——, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. 50

[25] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 746–751. 50

[26] F. L. de la Mora and S. Nadi, "Which library should i use?: A metric-based comparison of software libraries," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ACM, 2018, pp. 37–40. 8

[27] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, "How can i use this method?" In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, IEEE, vol. 1, 2015, pp. 880–890. 9

[28] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography APIs?" In *Proceedings of the 38th International Conference on Software Engineering*, ACM, 2016, pp. 935–946. 1

[29] 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 Institute of Technology, Tech. Rep*, 2012. 11

[30] *Parserannotator — stanford corenlp*, `https://stanfordnlp.github.io/CoreNLP/parse.html`, Accessed: 2018-07-13. 57

[31] *Parsing - parse usable street address, city, state, zip from a string - stack overflow*, `https://stackoverflow.com/questions/16413/parse-usable-street-address-city-state-zip-from-a-string/16446`, Accessed: 2018-08-1. 55

[32] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Software Engineering (ICSE), 2013 35th International Conference on*, IEEE, 2013, pp. 1295–1298. 9, 11

[33] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining stackoverflow to turn the ide into a self-confident programming prompter," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 2014, pp. 102–111. 9, 11

[34] M. P. Robillard, "What makes APIs hard to learn? answers from developers," *IEEE software*, vol. 26, no. 6, 2009. 1

[35] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011. 1

[36] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, *et al.*, "On-demand developer documentation," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, IEEE, 2017, pp. 479–483. 1

[37] *Search — github developer guide*, `https://developer.github.com/v3/search/`, Accessed: 2018-02-28. 28

[38] *Stack exchange data dump : Stack exchange, inc. : Free download, borrow, and streaming : Internet archive*, `https://archive.org/details/stackexchange`, Accessed: 2018-07-13. 56

[39] *Stack exchange data explorer*, `https://data.stackexchange.com/`, Accessed: 2018-06-18. 2, 13

[40] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding API components and examples," in *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, IEEE, 2006, pp. 195–202. 8

[41] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving API documentation using usage information," in *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, ACM, 2009, pp. 4429–4434. 1, 10

[42] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live API documentation," in *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 643–652. 10, 11, 48

[43] F. Thung, "API recommendation system for software development," in *Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference on*, IEEE, 2016, pp. 896–899. 9

[44] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, IEEE, 2013, pp. 182–191. 9

[45] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of API methods from feature requests," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, IEEE Press, 2013, pp. 290–300. 9

[46] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from stack overflow," in *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, IEEE, 2016, pp. 392–403. 1, 2, 10, 11, 21–23, 46

[47] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 565–581, 2015. 1, 2, 10, 14–17, 23

[48] W. M. Trochim and J. P. Donnelly, "Research methods knowledge base," 2001. 52

[49] *Ualberta-smr/benyamin-task-oriented-documentation: This repository holds the code and documents required to reproduce the outputs of task-oriented library documentation.* https://github.com/ualberta-smr/TaskOrientedDocumentation, Accessed: 2018-09-10. 6

[50] G. Uddin and M. P. Robillard, "How API documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, Jul. 2015, ISSN: 0740-7459. DOI: 10.1109/MS.2014.80. 1

[51] G. Uddin and F. Khomh, "Automatic summarization of API reviews," in *Automated Software Engineering (ASE), 2017 32nd IEEE/ACM International Conference on*, IEEE, 2017, pp. 159–170. 8

[52] D. Wang and T. Li, "Document update summarization using incremental hierarchical clustering," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ACM, 2010, pp. 279–288. 46

[53] *What are tags, and how should i use them? - help center - stack overflow*, https://stackoverflow.com/help/tagging, Accessed: 2018-06-18. 12

[54] *Windows - (uwp) webclient and downloading data from url in - stack overflow*, https://stackoverflow.com/questions/33123082/uwp-webclient-and-downloading-data-from-url-in, Accessed: 2018-08-6. 55

Appendix A

Sruvey Recruitment Emails

A.1 Recruitment Strategy 1

The following emails was sent out to undergraduate and graduate students of our department.

> Hello,
>
> I'm a Master's student working under supervision of Dr. Sarah Nadi. We are evaluating our research on building task-oriented summaries of Java software libraries through an online survey. If you have worked with Java before, we would like to invite you to participate.
>
> The survey can be found at `LINK-TO-SURVEY`. It will remain open until January 31st. This survey should take no more than 10 minutes of your time.
>
> Thank you for your participation. If you have any comments or questions, please do not hesitate to contact us:
>
> Benyamin Noori (email: bnoori@ualberta.ca)
>
> Sarah Nadi (email: nadi@ualberta.ca, website: http://www.sarahnadi.org)
>
> The plan for this study has been reviewed for its adherence to ethical guidelines by a Research Ethics Board at the University of Alberta. For questions regarding participant rights and ethical

conduct of research, contact the Research Ethics Office at (+1)-(780)-492-2615.

A.2 Recruitment Strategy 2

This recruitment strategy consisted of two parts. In the first part, we sent out the following emails to developers who had contributed to the libraries we selected for our experiement:

> Dear `USERNAME`,
>
> We are a group of researchers from the Department of Computing Science at the University of Alberta, Canada, who work on developing tools and methodologies to help software developers use Java libraries more easily and correctly. We have developed a new technique for building task-oriented summaries of software libraries.
>
> We noticed that you have contributed to a repository which belongs to one of the libraries included in our work, `LIBRARY-NAME`. Accordingly, we would like to invite you to participate in a short survey about the task-oriented library summaries that we built. The survey can be found in this link `LINK-TO-SURVEY`. We would appreciate it if you can fill out the survey as soon as possible, but note that it will remain open until February 28th. The survey should take no more than 5-10 minutes of your time.
>
> Note that the survey is completely anonymous. We only record the fact that you have used one of the libraries in our data, but do not record anything about your identity or activities. More information about how the data we collect is used can be found on the information page of the survey.
>
> Thank you for your time.
>
> Benyamin Noori (email: bnoori@ualberta.ca)
>
> Sarah Nadi (email: nadi@ualberta.ca, website: http://www.sarahnadi.org)

In the second part, we asked developers who had used the libraries in our experiement in one of their repositories to participate in the survey:

> Dear `USERNAME`,
>
> We are a group of researchers from the Department of Computing Science at the University of Alberta, Canada, who work on developing tools and methodologies to help software developers use Java libraries more easily and correctly. We have developed a new technique for building task-oriented summaries of software libraries.
>
> We noticed that you have committed to a source file that uses the jaxb library in the Github repository `REPOSITORY NAME`, which is one of the libraries included in our work.
>
> Accordingly, we would like to invite you to participate in a short survey about the task-oriented library summaries that we built. The survey can be found in this link `LINK-TO-SURVEY`. We would appreciate it if you can fill out the survey as soon as possible, but note that it will remain open until February 28th. The survey should take no more than 5-10 minutes of your time. Note that the survey is completely anonymous. We only record the fact that you have used one of the libraries in our data, but do not record anything about your identity or activities. More information about how the data we collect is used can be found on the information page of the survey.
>
> Thank you for your time.
>
> Benyamin Noori (email: bnoori@ualberta.ca)
>
> Sarah Nadi (email: nadi@ualberta.ca, website: http://www.sarahnadi.org)

Appendix B

Survey Results

In this appendix, we provide the details of responses we received in our survey. For presentation purposes, we replaced the professions of the participants with letters. The mappings of letters to professions can be seen in table B.

Category	Description
A	UGrad Comp Sci
B	Grad Comp Sci
C	Industrial Dev
D	Academic Researcher
E	Freelance Dev

Table B.1: Mappings between letters and participants' occupations.

The following table shows the details of every response we received to our survey.

Stage 1			Stage 2									Stage 3				
1.a	1.b	1.c	2.task-1			2.task-2			2.task-3			3.a	3.b	3.c	3.d	3.e
			Task	Snippets	Insights	Task	Snippets	Insights	Task	Snippets	Insights					
A	0	1-2	Yes	3,4,2	1							3	3	1	3	4
A	0	1-2	Yes	4,2								4	3	1	3	4
B	0	< 1	Yes	5,3,3	5,4,4,4							3	4	4	5	5
B	0	2-5	Yes	2,3								5	3	1	3	3
B	0	2-5	IDK	3,4	2,2,4							4	4	3	3	3
B	0	2-5	Yes	5,2,1	5,2,4,4	Yes	5	5,3	Yes	5,5,3	3,4,4,4,3,2	4	5	4	3	5
B	0	1-2	IDK	1,5								5	5	1	3	3
A	0	2-5	Yes	4,2,2		IDK	4,2	3,4				3	3	4	5	3
A	0	2-5	Yes	5,1,1	4,4,5							4	5	4	4	5
B	0	1-2	IDK	2,1,4	2							2	3	3	1	3
B	0	1-2	Yes	5	5,5	Yes	5,5,1	5				5	5	4	5	5
B	0	2-5	Yes	5	2,2							3	5	2	4	2
B	20+	2-5	Yes	5,4,3								2	4	3	4	3
B	1-5	2-5	Yes	1,1,5	4,1,3,2,1,5							3	5	3	1	4
B	0	6-10	IDK	4,3,5								4	4	5	4	5
B	1-5	2-5	Yes	3,3,3	2,3,5	Yes	3,5,1	1,2	Yes	3,4	4,1	3	4	3	2	4
B	0	2-5	Yes	4,5,5	3							4	5	2	3	4
A	0	< 1	No	4,3,4	4,5,3							3	3	4	3	4
B	0	1-2	Yes	5	4,4							4	5	4	4	4

B	0	1-2	Yes	5,3,1	3,5,1,1,2,2	IDK	3,3	1,1				3	3	4	3	3
B	1-5	2-5	Yes	5,4		IDK	4,4,4		IDK	5,5	5,3,2,3,2,3,1	3	4	3	4	4
B	0	2-5	IDK	3,1	1,2,3	Yes	3		IDK	5	3,1	3	4	3	2	3
B	0	1-2	Yes	5,3,3	5,3,4,4	Yes	4	3,4	Yes	4,4,3	2,4,3,4,4,3	3	5	4	3	4
B	0	2-5	Yes	5	1,1	IDK	4,4,1	1				3	4	1	3	3
B	0	2-5	Yes	3,3,4		IDK	4,2	2,1				2	2	2	2	3
B	0	2-5	Yes	5,4,3	3	Yes	5,3	4,3,3,4	Yes	5,5	5	4	5	5	4	4
B	1-5	11+	Yes	5,2,4	2,5,4,4,3,5	Yes	4,5	3,1	Yes	4,4,5	4,4,5,2,4,3,3,1,4,2	3	5	4	4	3
B	1-5	11+	Yes	5,5,5	4,5,4							5	5	5	5	4
A	20+	2-5	Yes	5,4,3	4							4	5	4	4	4
D	1-5	11+	IDK	5,2,3	3,2,2							3	4	5	3	4
C	1-5	6-10	Yes	2,5,5	3,5,3							4	4	4	4	4
C	20+	11+	Yes	5,3,5								3	4	3	4	4
C	0	1-2	IDK	4,4,4		IDK	3,2	4,3				5	4	4	5	3
D	1-5	6-10	Yes	5,1,5	3,4,1,2,2,5							1	5	4	3	1
E	1-5	11+	No	3,3	5,3,3	Yes	5					3	4	4	3	4
C	1-5	2-5	Yes	3,3,4	5							5	3	4	2	4
E	1-5	11+	Yes	5,4,4	5	Yes	1,4	3,3,2,4	Yes	3,3	3	3	2	3	1	4
C	1-5	11+	Yes	5	4,4	No	5,3,3	3	No	4,3,2	2,5,3	3	4	3	4	3
A	1-5	1-2	Yes	3,4,4	4,4,4							4	4	4	4	4
A	1-5	1-2	Yes	3,4,4	4,4,4							4	4	4	4	4
C	1-5	11+	Yes	5,3,5	4,5,4,5,4,5							4	4	4	4	5
E	1-5	11+	NOT	1,1,1	1,4,2	Yes	2,5,1	1,2	Yes	2,4	1,2	1	2	2	1	5
C	1-5	2-5	Yes	5,5,5	3,4,5	Yes	5,5,5	5,5	Yes	5,4	4,4	4	4	4	4	4
C	1-5	2-5	Yes	1,1,5	4							3	3	3	3	4

C	10-20	2-5	Yes	2	2,2	Yes	5,3,3	5				3	3	2	4	3
E	10-20	2-5	Yes	3,3,1	4,3,2,2,4,5							2	3	4	2	2
C	1-5	6-10	Yes	5,4,1	3,5,1							4	4	3	1	2
C	1-5	2-5	Yes	4,4	4,4,4							4	4	5	4	4
C	20+	6-10	Yes	5,5,5								5	5	5	5	5
C	1-5	6-10	Yes	3,3,5	1,4,4	Yes	3,5,4	2,4				4	4	4	4	4
C	1-5	2-5	IDK	3,2,5	4							2	4	5	2	3
A	1-5	1-2	Yes	5,2,4								5	4	3	3	4
E	1-5	11+	Yes	4,3,4	5	Yes	4,3	3,3,1,3	Yes	4,4	1	4	2	2	3	4
E	10-20	2-5	Yes	1,4,1	2	Yes	5		No	3	2	4	4	3	5	5
B	1-5	2-5	Yes	5,1,2		Yes	4,2	2,2				3	3	3	5	5
B	20+	2-5	No	1,1,5	2,5,4							4	4	3	2	3
C	1-5	1-2	Yes	4,4,5	4,5,5							4	5	3	4	5
C	1-5	11+	No	4,4	3,4,2	Yes	5		No	4	3,3	2	4	3	1	5
C	1-5	11+	Yes	5,4,5	5							4	5	5	3	4
C	1-5	6-10	Yes	5	5,4	No	1,1,1	1	Yes	4,4,4	3,5,3	4	4	4	4	4
C	10-20	11+	Yes	4,4,1	5,5,3,2,1,2	Yes	3,5	3,1				5	5	4	4	4
A	1-5	< 1	Yes	4,4,4	4,4,3	Yes	4,5,2	1,3				4	5	3	3	4
C	1-5	11+	Yes	1,5		Yes	5,5,5		No	1,3	1,1,1,1,1,1,1	1	1	1	1	1
C	1-5	2-5	No	4,1	5,5,4	Yes	4					1	1	1	1	1
A	1-5	2-5	Yes	5,3,3	3,4,5,4,4,5							2	4	4	4	5
C	1-5	11+	Yes	5,2,3	1							3	3	1	2	3
C	1-5	2-5	Yes	4,3,3	1							4	2	1	2	1
A	1-5	1-2	No	5,5,1	5							4	4	4	1	2
C	1-5	1-2	Yes	1,5,1	2	No	1					4	5	3	4	4

Also, the open-ended comments written by the participants are detailed in the following table.

1	The task description is sometimes an unclear sentence or phrase. In some cases I should go to the stack overflow page to understand the task completely. It would be better to make a self contained description for the task.
2	The answers given on stackoverflow say how to do the task, but not necessarily given the library, and they don't give any written justification about why they don't use the library. However, maybe people would have given better answers if the question was more descriptive. Another thing that would have made the answers better would have been breaking the code up into snippets and describing each subtask. In my experience, when people are browsing SO, if they see a question/answer in which nothing stands out them they will immediately navigate to the next link. The user wants to understand the question/problem quickly and without thinking too hard - so it needs to be as human-readable as possible while also providing any relevant code examples.
3	To avoid spending time on the whole documentation, finding a list of tasks (from basic to advanced) that can be accomplished using a library is the first thing that I would be looking for when I need to work with the library.
4	I don't understand the insight sentences unless I read the stack overflow page.
5	The list of examples for the use of Bouncy Castle is helpful. However, I often need information to work with the native BC API. Documentation is more often oriented with JCE API.
6	Testng is not anymore relevant, many of its advantages are now fully handled by junit4/5 and extra-runners. Most project don't use testng anymore.
7	I need more completed demos that can run as an application
8	A number of the code blocks provided didn't seem to be of the standard I would want to use as examples in documentation.
9	The insight sentences don't have much detail are are of very limited value without context.

10	I'm not sure if this would provide more direct value in working with a library than: Javadocs, Google or Communities (eg: mailing list, stack overflow, irc).
11	I can see far more value in this work for being able to match what libraries could accomplish a particular task than the other way around. Especially combined with ranking libraries suitable for a task of collection of tasks.
12	Interesting work. Please keep us posted on your progress.
13	The code snippets provided did not use bouncycastle. They did not use any classes in org.bouncycastle.
14	As I wrote to you in email if question is unrelated to library then no other questions should be mandatory and must be skipped. Answered '1' is such case.
15	Some questions are quite complicated and task description is not enough, though link to SO helps. Same for snippets-answers.
16	The official documentation was more helpful to me than Stack overflow. But, Stack Overflow helped me to work through some of the bugs in my code during the execution of the library.