TaskNav: Task-based Navigation of Software Documentation

Christoph Treude*, Mathieu Sicard†, Marc Klocke†, and Martin Robillard†
*Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, RN, Brazil
Email: ctreude@dimap.ufrn.br
†School of Computer Science, McGill University, Montréal, QC, Canada
Email: {mathieu.sicard, marc.klocke}@mail.mcgill.ca, martin@cs.mcgill.ca

***Abstract*—To help developers navigate documentation, we introduce TaskNav, a tool that automatically discovers and indexes task descriptions in software documentation. With TaskNav, we conceptualize tasks as specific programming actions that have been described in the documentation. TaskNav presents these extracted task descriptions along with concepts, code elements, and section headers in an auto-complete search interface. Our preliminary evaluation indicates that search results identified through extracted task descriptions are more helpful to developers than those found through other means, and that they help bridge the gap between documentation structure and the information needs of software developers.**
Video: https://www.youtube.com/watch?v=opnGYmMGnqY

I. Introduction and Motivation

Software documentation is disseminated in many different forms, written by different individuals for different purposes [16]. While the emergence of social media has promoted the availability of software documentation for a wide variety of topics [13], there is often a mismatch between the needs of documentation users and the knowledge provided by documentation writers. Although documentation often follows a hierarchical structure with sections and subsections, this organization can only enable effective documentation use if section headers are adequate cues for the information needs of the users.

Search engines are insufficient for enabling effective navigation of software documentation because they require users to use search terms that match the vocabulary used by the documentation writers. To close this vocabulary gap [9], search engines often employ an auto-complete feature, based on query stream mining [1], ontologies [9], or concepts extracted from the corpus using statistical methods [2]. None of these approaches are applicable to specialized search systems for software documentation as corpora are too small to train statistical models and query streams are not available or not representative of specific search needs in a constantly evolving documentation landscape.

To overcome these issues, we introduce TaskNav, a task-based search engine for software documentation. TaskNav automatically analyzes a documentation corpus (typically an online tutorial) and detects every passage that describes how to accomplish some programming task. Our task-detection approach relies on natural language processing (NLP) techniques. We define a task as a specific programming action that has been described in the documentation. For example, a task for a developer in Java could be *"get iterator for collection"*. TaskNav surfaces these task descriptions in an interactive auto-complete search interface, along with automatically extracted concepts, code elements, and section titles (see Figure 1).

TaskNav can automatically analyze and index any documentation corpus based on a starting URL and some configuration parameters, such as which HTML tags should be ignored. Documentation users can benefit from TaskNav by taking advantage of the task-based navigation offered by the auto-complete search interface. For documentation writers, TaskNav provides analytics that show how documentation is used (e.g., top queries, most frequently read documents, and unsuccessful searches). Researchers can benefit from the data accumulated by TaskNav's logging mechanism as it provides detailed data on how software developers search and use software documentation.

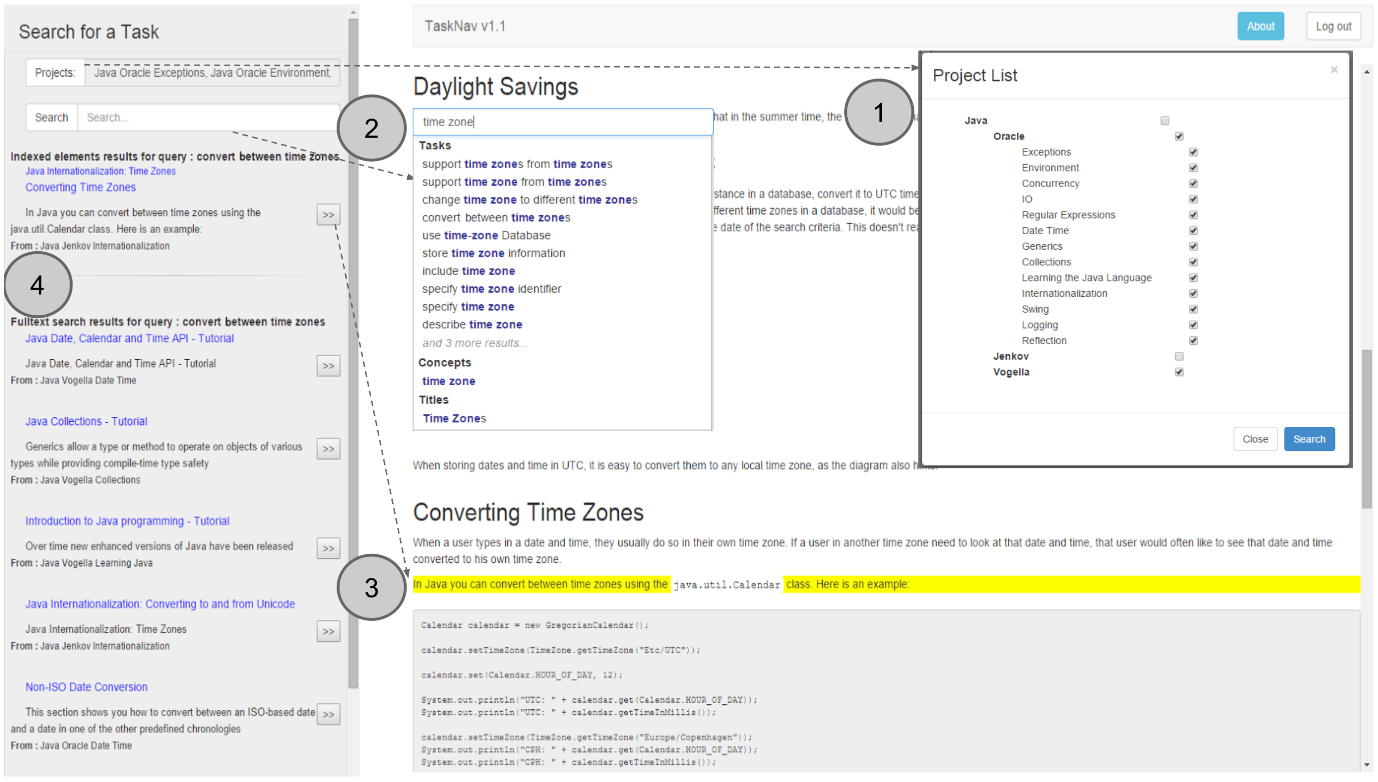
II. TaskNav

In this section, we describe TaskNav's task extraction algorithm as well as the interactive auto-complete interface.

Task Extraction. We conceptualize tasks in software documentation as verbs associated with a direct object and/or a prepositional phrase, such as *"get iterator"*, *"get iterator for collection"*, or *"add to collection"*. To extract task descriptions from software documentation, we make use of the grammatical dependencies between words, as detected by the Stanford NLP parser [8].

In the first step, we remove markup from the HTML files and preprocess the resulting text files to account for the unique characteristics of software documentation not found in other texts, such as the systematic use of incomplete sentences and the presence of code terms. In particular, we prefix sentences that start with a verb in present tense, third person singular, such as *"returns"* or *"computes"*, with the word *"this"* to ensure correct parsing of partial sentences, such as *"Returns the next page number"*. In addition, we prefix sentences that start with a verb in present participle or gerund, such as *"adding"* or *"removing"*, immediately followed by a noun,

This work was conducted while Christoph Treude was a postdoctoral researcher at McGill University.



Search for a Task

Projects: Java Oracle Exceptions, Java Oracle Environment,

Search Search...

Indexed elements results for query : convert between time zones
Java Internationalization: Time Zones
Converting Time Zones
In Java you can convert between time zones using the java.util.Calendar class. Here is an example:
From : Java Jenkov Internationalization

Fulltext search results for query : convert between time zones
Java Date, Calendar and Time API - Tutorial
Java Date, Calendar and Time API - Tutorial
From : Java Vogella Date Time

Java Collections - Tutorial
Generics allow a type or method to operate on objects of various types while providing compile-time type safety
From : Java Vogella Collections

Introduction to Java programming - Tutorial
Over time new enhanced versions of Java have been released
From : Java Vogella Learning Java

Java Internationalization: Converting to and from Unicode
Java Internationalization: Time Zones
From : Java Jenkov Internationalization

Non-ISO Date Conversion
This section shows you how to convert between an ISO-based date and a date in one of the other predefined chronologies
From : Java Oracle Date Time

Daylight Savings

time zone

Tasks
support **time zone**s from **time zone**s
support **time zone** from **time zone**s
change **time zone** to different **time zone**s
convert between **time zone**s
use **time-zone** Database
store **time zone** information
include **time zone**
specify **time zone** identifier
specify **time zone**
describe **time zone**
and 3 more results...
Concepts
time zone
Titles
Time Zones

hat in the summer time, the
stance in a database, convert it to UTC time
fferent time zones in a database, it would be
date of the search criteria. This doesn't rea

When storing dates and time in UTC, it is easy to convert them to any local time zone, as the diagram also h

Project List

Java	☐
Oracle	✓
Exceptions	✓
Environment	✓
Concurrency	✓
IO	✓
Regular Expressions	✓
Date Time	✓
Generics	✓
Collections	✓
Learning the Java Language	✓
Internationalization	✓
Swing	✓
Logging	✓
Reflection	✓
Jenkov	☐
Vogella	✓

Converting Time Zones

When a user types in a date and time, they usually do so in their own time zone. If a user in another time zone need to look at that date and time, that user would often like to see that date and time converted to his own time zone.

In Java you can convert between time zones using the `java.util.Calendar` class. Here is an example:

```
Calendar calendar = new GregorianCalendar();

calendar.setTimeZone(TimeZone.getTimeZone("Etc/UTC"));

calendar.set(Calendar.HOUR_OF_DAY, 12);

System.out.println("UTC: " + calendar.get(Calendar.HOUR_OF_DAY));
System.out.println("UTC: " + calendar.getTimeInMillis());

calendar.setTimeZone(TimeZone.getTimeZone("Europe/Copenhagen"));
System.out.println("CPH: " + calendar.get(Calendar.HOUR_OF_DAY));
System.out.println("CPH: " + calendar.getTimeInMillis());
```

Fig. 1. TASKNAV screenshot. (1) Project selection, (2) auto-complete, (3) highlighted search result, (4) full-text search results

with the word *"for"* to ensure correct parsing of partial sentences, such as *"Displaying data from another source"*.

We further configure the NLP parser to automatically tag all code elements as nouns, as suggested by Thummalapenta et al. [14]. In addition to code terms explicitly tagged with `tt` or `code` tags in the original HTML, all words that match one of about 30 regular expressions are masked as code terms. The regular expressions are available in our online appendix.[1] The resulting sentences are then parsed using the Stanford NLP toolkit.

To extract task descriptions, TASKNAV makes use of the grammatical dependencies—relations between words in a sentence—identified by the Stanford NLP toolkit. Because tasks can be described in different grammatical ways (e.g., *"returning an iterator"*, *"return iterator"*, *"iterator returned"*, and *"iterator is returned"*), dependencies between words in active and passive voice have to be considered. In addition, context might be important, e.g., whether an iterator is returned or whether the documentation instructs the user to *"not return iterator"*. Furthermore, the iterator might be specified using additional words, such as *"list iterator"*, and prepositional phrases might make the task description more specific, such as *"return iterator of collection"*. Our approach also detects multiple task descriptions if tasks are intertwined using conjunctions. For example, the partial sentence *"return list or iterator"* would result in two task descriptions: *"return list"* and *"return iterator"*.

[1]http://cs.mcgill.ca/~swevo/tasknavigator/

To ensure that general verbs, such as *"contain"*, are not used to define a task, we filter task descriptions to only include those for which the verb is a programming action. A handcrafted list of about 200 programming actions is available in our online appendix. Similarly, we handcrafted a list of generic objects, such as *"this"* and *"it"*, to avoid extracting task descriptions such as *"return this"*. Finally, task descriptions are normalized by using the base form of the verb (e.g., *"return"* instead of *"returning"*).

In addition to task descriptions, TASKNAV extracts concepts using a well-recognized technique for collocation detection [7, Chapter 5], code elements using HTML tags and regular expressions, and section titles using HTML header tags.

User Interface. TASKNAV suggests the extracted task descriptions along with concepts, code elements, and section titles in an interactive auto-complete search interface. Figure 1 outlines the search process enabled by TASKNAV. First, the user selects which projects to include in a search (1). Projects are organized in a hierarchy where the first level represents the technology (e.g., Java), the second level represents the author or web domain of the documentation (e.g., Oracle), and the third level represents the subject (e.g., JUnit). These descriptors are specified whenever a new project is added to TASKNAV. Multiple projects can be selected for any search, and TASKNAV remembers a user's project selection.

When the user starts typing a search query (2), an auto-complete list opens and shows task descriptions, concepts, section titles, and code elements that contain all words that have been typed so far. The words do not have to appear in the order in which they were typed. Once the user runs the search query, results are presented on the left side of the screen. For each result, the title (and possibly sub-title) of the corresponding section is shown as a link, and the paragraph that matched the query is displayed underneath the title. When the user selects a result by clicking on either the title or the button next to the matching paragraph, the corresponding document is opened on the right side of the screen. The paragraph that matched the query is highlighted, and the document is automatically scrolled to that paragraph (3). Underneath the search results based on its index, TASKNAV displays search results based on traditional full text search using Apache Lucene (4). In addition to the search functionality, TASKNAV includes user interfaces for indexing new documentation and for analytics.

III. EXAMPLE SCENARIOS

In this section, we describe two example scenarios to highlight TASKNAV's functionality, using the Java Tutorials from Oracle as documentation corpus.

Converting Time Zones. We first describe the scenario depicted in Figure 1. The user has to manage different time zones in a Java program and uses TASKNAV for help. After typing *"time zone"* into TASKNAV's search field, several task descriptions, concepts, and section titles are suggested. The user realizes that the documentation writers have used different vocabulary to refer to the task of converting between time zones, such as *"change time zone to different time zones"* and *"convert between time zones"*. This is also a good example of the variety of grammatical structures that TASKNAV supports for task descriptions: *"convert between time zones"* consists of a verb and a prepositional phrase, *"change time zone to different time zones"* consists of a verb, a direct object, and a prepositional phrase, and another task description, such as *"specify time zone"*, consists of a verb and a direct object.

After choosing the query *"convert between time zones"* from auto-complete, the user selects the one search result that was generated based on TASKNAV's index, as shown in Figure 1-3. The corresponding web page is automatically scrolled to the paragraph from which the task description was extracted, and the paragraph is highlighted: *"In Java you can convert between time zones using the* `java.util.Calendar` *class. Here is an example"*. Now the user can continue by investigating the code snippet following the paragraph.

Avoiding Thread Interference. For the second scenario, we assume that the user wants to solve a problem concerning thread interference. After typing *"thread in"* into the search field, the auto-complete suggestions shown in Table I appear. Just from looking at the auto-complete suggestions, the user now knows that avoiding thread interference is described using the verb *"prevent"* in the documentation. After selecting the corresponding entry from auto-complete, three search results are displayed since the task *"prevent thread interference"* is described more than once in the documentation: The section on *"Synchronization"* refers to its own subsection *"Synchronized Methods"* using the sentence *"Synchronized Methods describes a simple idiom that can effectively prevent thread interference and memory consistency errors."* In that section, the sentence *"Synchronized methods enable a simple strategy for preventing thread interference and memory consistency errors"* contains the task description, whereas the tutorial on *"Atomic Variables"* contains the sentence *"Replacing the* `int` *field with an* `AtomicInteger` *allows us to prevent thread interference without resorting to synchronization, as in* `AtomicCounter`*"*, followed by a code snippet. TASKNAV associated all three sentences with the task description *"prevent thread interference"*.

TABLE I
AUTO-COMPLETE SUGGESTIONS AFTER TYPING *"thread in"*

Tasks
introduce thread interference
prevent thread interference
use ThreadLocalRandom instead of Math.random()
run threads in program
prevent thread interference without resorting
use background thread instead of event-dispatching thread
create instance of Thread
run code in thread
execute in thread
introduce thread contention
Concepts
initial thread
Titles
Threads in Applets
Initial Threads
Threads

IV. PRELIMINARY EVALUATION

We deployed a prototype of the TASKNAV web application at a web development company in Montréal, Canada. During a two-week field study in which six professional developers (P1–P6) used the tool as part of their normal ongoing work, we recorded all their interactions and we asked them *"Was this what you were looking for?"* on every other click on a link for a search result, giving *"yes"* and *"no"* as answer options in a pop-up window.

Table II shows the results of the field study. For each participant P1–P6, it shows the number of queries entered by the participant. The last columns show the number of clicks on search results along with how often the answer to *"Was this what you were looking for?"* was *"yes"* or *"no"*, respectively. A total of 83 search results were selected during the field study, and the participants answered whether the result was what they were looking for in 37 cases. Fifteen of the answers were positive and 19 were negative. The results divided up by the different indexed elements clearly indicate the usefulness of task descriptions: out of 10 answers about task-related search results, 8 were positive, while most answers about search results related to code elements and section titles were negative. This difference is statistically significant (Fisher's exact test, $p < 0.05$). The difference between results derived from task descriptions and section titles is particularly noteworthy: section titles are meant to help developers navigate the documentation, yet the corresponding results received overwhelmingly negative feedback, while task descriptions stood out as the most useful way to navigate

TABLE II
RESULTS OF THE FIELD STUDY

	queries	**clicks (relevant / not relevant)**			
		tasks	**concepts**	**code**	**titles**
P1	20	3 (1/0)	0 (0/0)	2 (0/1)	4 (0/1)
P2	19	6 (1/1)	0 (0/0)	8 (1/2)	3 (2/0)
P3	11	5 (2/0)	0 (0/0)	8 (2/1)	3 (0/2)
P4	40	9 (3/1)	0 (0/0)	11 (0/6)	12 (0/4)
P5	12	1 (1/0)	1 (1/0)	3 (0/0)	0 (0/0)
P6	4	1 (0/0)	0 (0/0)	3 (1/0)	0 (0/0)
sum	106	25 (8/2)	1 (1/0)	35 (4/10)	22 (2/7)

documentation. The results also indicate that concepts—used in other domains for populating auto-complete fields [2]—were hardly considered by the participants in the field study.

Additional details of the evaluation are available elsewhere [15]. TASKNAV is currently deployed at McGill University and in use by undergraduate students.

V. RELATED WORK

While information extraction in other domains is often limited to detecting concepts, our focus on tasks was motivated by previous work on the importance of tasks in software development [6]. Task extraction from natural language documents has been the object of research in areas outside of software engineering. Scerri et al. presented a technology for the automatic classification of email action items based on a model that considers five linguistic, grammatical and syntactical features. Their model is rich enough to capture action-object tuples, such as *"request data"* [10]. Kalia et al. went a step further to present an approach for automatically identifying task creation, delegation, completion, and cancellation in email and chat conversations, based on NLP techniques and machine learning. They distinguished between four types of tasks: create, delegate, discharge, and cancel [5]. Compared to our approach, these models do not allow for more complex tasks such as *"get iterator for collection"*, but they are richer in terms of who does an action and whether this action is requested, suggested, or demanded—which is less relevant in software documentation.

TASKNAV is also related to textual feature location, and in particular the work by Shepherd et al. [11], [12]. Similar to our work, their approach is based on the notion that actions in software development can be represented by verbs and nouns correspond to objects. Their tool, Find-Concept, allows developers to create source code queries consisting of a verb and a direct object. Find-Concept then expands the queries using NLP and knowledge of the terms used within the source code to recommend new queries. Our work differs from Find-Concept and other query expansion tools, such as the work by Hill et al. [4] or Haiduc et al. [3], in several ways: In query expansion tools, the initial query needs to be a complete query. TASKNAV only needs two characters to trigger auto-complete suggestions, thus allowing developers to use the system even if they do not know how to phrase the complete query yet. Our task descriptions are also more precise by incorporating prepositions and prepositional objects in addition to verbs and direct objects.

VI. CONCLUSION

To help bridge the gap between the information needs of software developers and the structure of existing documentation, we have developed TASKNAV, a tool that automatically extracts task descriptions from software documentation and suggests them in an auto-complete interface. Our preliminary evaluation indicates that task descriptions can be extracted from software documentation automatically, and that they can help developers navigate software documentation.

TASKNAV is now deployed and in operation at McGill University, and it will be released to the general public in early 2015.

REFERENCES

[1] M. Barouni-Ebrahimi and A. A. Ghorbani. On query completion in web search engines based on query stream mining. In *Proc. of the Int'l. Conf. on Web Intelligence*, pages 317–320, 2007.

[2] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *Proc. of the 34th Int'l. Conf. on Research and Development in Information Retrieval*, pages 795–804, 2011.

[3] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *Proc. of the 35th Int'l. Conf. on Software Engineering*, pages 842–851, 2013.

[4] E. Hill, L. Pollock, and K. Vijay-Shanker. Automatically capturing source code context of NL-queries for software maintenance and reuse. In *Proc. of the 31st Int'l. Conf. on Software Engineering*, pages 232–242, 2009.

[5] A. Kalia, H. R. M. Nezhad, C. Bartolini, and M. Singh. Identifying business tasks and commitments from email and chat conversations. Technical Report HPL-2013-4, HP Laboratories, 2013.

[6] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *Proc. of the 14th Int'l. Symp. on the Foundations of Software Engineering*, pages 1–11, 2006.

[7] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[8] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[9] P. Mika, E. Meij, and H. Zaragoza. Investigating the semantic gap through query log analysis. In *Proc. of the 8th Int'l. Semantic Web Conference*, pages 441–455, 2009.

[10] S. Scerri, G. Gossen, B. Davis, and S. Handschuh. Classifying action items for semantic email. In *Proc. of the 7th Int'l. Conf. of Language Resources and Evaluation*, pages 3324–3330, 2010.

[11] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz. Sando: An extensible local code search framework. In *Proc. of the 20th Int'l. Symp. on the Foundations of Software Engineering*, pages 15:1–15:2, 2012.

[12] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proc. of the 6th Int'l. Conf. on Aspect-oriented Software Development*, pages 212–224, 2007.

[13] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng. The impact of social media on software engineering practices and tools. In *Proc. of the Workshop on Future of Software Engineering Research*, pages 359–364, 2010.

[14] S. Thummalapenta, S. Sinha, D. Mukherjee, and S. Chandra. Automating test automation. Technical Report RI11014, IBM Research Division, 2011.

[15] C. Treude, M. Robillard, and B. Dagenais. Extracting development tasks to navigate software documentation. *IEEE Trans. on Software Engineering*, 2015. To appear.

[16] C. Treude and M.-A. Storey. Effective communication of software development knowledge through community portals. In *Proc. of the 8th joint meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering*, pages 91–101, 2011.