How to Formulate Specific How-To Questions in Software Development?

Mingwei Liu*
Fudan University
China

Xin Peng*
Fudan University
China

Andrian Marcus
The University of Texas at Dallas
USA

Christoph Treude
The University of Melbourne
Australia

Jiazhan Xie*
Fudan University
China

Huanjun Xu*
Fudan University
China

Yanjun Yang*
Fudan University
China

ABSTRACT

Developers often ask how-to questions using search engines, technical Q&A communities, and interactive Q&A systems to seek help for specific programming tasks. However, they often do not formulate the questions in a specific way, making it hard for the systems to return the best answers. We propose an approach (TaskKG4Q) that interactively helps developers formulate a programming related how-to question. TaskKG4Q is using a programming task knowledge graph (task KG in short) mined from Stack Overflow questions, which provides a hierarchical conceptual structure for tasks in terms of [*actions*], [*objects*], and [*constraints*]. An empirical evaluation of the intrinsic quality of the task KG revealed that 75.0% of the annotated questions in the task KG are correct. The comparison between TaskKG4Q and two baselines revealed that TaskKG4Q can help developers formulate more specific how-to questions. More so, an empirical study with novice programmers revealed that they write more effective questions for finding answers to their programming tasks on Stack Overflow.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; **Documentation**.

KEYWORDS

Programming Task, Query Formulation, Stack Overflow

ACM Reference Format:
Mingwei Liu, Xin Peng, Andrian Marcus, Christoph Treude, Jiazhan Xie, Huanjun Xu, and Yanjun Yang. 2022. How to Formulate Specific How-To Questions in Software Development?. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3540250.3549160

*M. Liu, X. Peng, J. Xie, H. Xu and Y. Yang are with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China.

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3549160

1 INTRODUCTION

How-to questions are questions that ask for instructions [64], for example *"How to read a json file in java"*. Developers often ask how-to questions using search engines [34, 36], technical Q&A sites [64], and interactive Q&A systems [12, 25, 63, 72], seeking help for their programming tasks. For example, Treude *et al.* [64] found that 39.22% of Stack Overflow (SO) questions are how-to questions, while other researchers have analyzed and categorized SO questions [14, 17, 18, 57, 64], with how-to questions consistently emerging as one of the most important categories, shared by all taxonomies.

Previous studies found that it is not easy for developers to formulate a good question [41, 48, 72]. Rahman *et al.* [51] conducted an empirical study on code search using Google and found that developers often miss important technical details (*e.g.,* programming languages or operating systems) in their initial queries. A programming task usually consists of essential elements such as [*actions*], [*objects*], and [*constraints*] [29, 65]. For example, in the task *"read a json file in java"*, *"read"* is the [*action*], *"json file"* is the [*object*], and *"in java"* is the [*constraint*]. For a how-to question, the key to whether it can be understood and answered correctly by a human or automated approaches is whether the question is specific enough, *i.e.,* the essential task elements in the question are completely and precisely expressed. For example, if the constraint *"in java"* is missing, one may get answers in Python; likewise, if the object is *"file"* or *"json"* instead of *"json file"*, one may get answers for reading an XML file or a JSON array. At the same time, a programming task can be expressed in many different ways, *e.g.,* by using different sentence patterns. Hence, developers may struggle to choose the best formulation and need to refine and revise their questions before getting the right answers. There are a total of 2,554,062 edit records for SO question titles and about 12% of the questions with the keyword *"how to"* in the title have title edits (as of 03/2021), underlining this phenomenon. More so, in a sample of 410 how-to questions with title edits, we found that task elements (*i.e.,* [*actions*], [*objects*], and [*constraints*]) were involved in 71% of

the edits (see Sec. 3), which indicates that developers may struggle to describe the task elements well enough the first time.

Software engineering researchers proposed several techniques for helping developers improve their questions or queries for different tasks, *e.g.,* bug localization and code search. Some approaches focus on automatically reformulating questions by expanding them with relevant terms [21–23, 26, 35, 42, 44, 45, 49, 52–54, 56, 60], while others help users improve their questions interactively [66, 73, 74]. For example, Zhang *et al.* [73] designed Chatbot4QR to help developers refine their SO queries by asking clarification questions based on tags from existing posts. However, it only helps adding details to the initial query written by the user and it cannot replace poorly chosen terms in the initial query.

One common aspect to these past research efforts is that they do not address how-to questions explicitly. We argue that, due to their prevalence, helping the formulation of how-to questions deserves specific solutions, rather than *"one size fits all"* approaches, and this is the main goal of this paper.

Our premise is that new how-to questions asked by developers are similar to existing task-related questions, hence the structure and content of these past questions can help formulate new ones. Based on this idea, we propose an interactive approach and tool (TaskKG4Q) that can help developers formulate specific how-to questions. TaskKG4Q uses a programming task knowledge graph (task KG in short), constructed from SO question titles. The task KG provides a hierarchical conceptual structure for tasks in terms of [*actions*], [*objects*], and [*constraints*]. Based on the task KG, TaskKG4Q recommends appropriate and specific [*actions*], [*objects*] and [*constraints*] to developers, when they formulate how-to questions, and also suggests details that may be missing.

Our task KG construction approach (Sec. 4) relies on a set of three linguistic patterns, we manually identified in high-quality SO questions. From here on, the approach is automated and it: (1) extracts a set of seed tasks from SO questions, matching these patterns; (2) incrementally and iteratively discovers additional tasks, using task extension mechanisms; and (3) annotates questions with the extracted task elements. The resulting task KG, includes tasks with new concepts and corresponding questions with new linguistic patterns, beyond those provided initially. Also, the task KG can evolve automatically, as more data becomes available.

We constructed a task KG using 257,430 SO questions and found that our approach can accurately (75.0% accuracy) extract tasks from questions that refer to those tasks (Sec. 6.1). To evaluate TaskKG4Q's usefulness we asked novices to complete four programming tasks and write questions with/without the help of TaskKG4Q during this process (Sec. 6.3). They were able to write more effective how-to questions for Stack Overflow using the tool, and deemed the tool easy to use.

In summary, the contributions of this paper are:

- A conceptual model for describing tasks, task-related elements, and relations between them.
- An incremental and iterative approach that mines a task KG from SO questions, by combining top-down and bottom-up task extension strategies.
- A task KG with 266,659 tasks mined from SO questions.
- A first-of-its-kind interactive tool that helps developers formulate specific how-to programming questions.

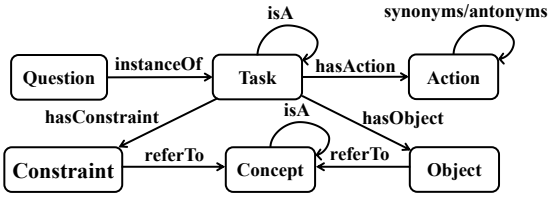

Figure 1: Main Concepts Schema

While our evaluation focused on writing how-to question on Stack Overflow and the task KG is built based on Stack Overflow, TaskKG4Q can be used independent of Stack Overflow, with any other programming related Q&A system or forum.

2 CONCEPTUAL MODEL

Fig. 1 shows the main concepts used in this paper and the relations between them. A [*task*] is a specific programming task that a developer wants to complete and is described by multiple elements: an [*action*], an [*object*] and multiple [*constraints*]. An [*action*] is a verb or verb phrase to describe the main action of the task. An [*object*] is a noun phrase representing the direct object of the action. A [*constraint*] is a prepositional phrase representing a constraint for completing the task. For example, the [*task*] *"read a json file in java"* includes an [*action*] (*i.e.,* read), an [*object*] (*i.e.,* json file) and a [*constraint*] (*i.e.,* in java). We record this [*task*] as the tuple `Task(read, json file, in java)` in the order of [*action*], [*object*] and [*constraints*].

As for relations, an [*object*] and a [*constraint*] may refer to the same concept, *e.g., "json file" and "from json file"*. There are hierarchical relations (*i.e., isA*) between concepts, *e.g.,* <java, *isA*, programming language>. This type of conceptual relation may come from existing knowledge bases (*e.g.,* general knowledge graph WikiData [11]). At the same time, there could be *synonym* (*e.g.,* remove and delete) or *antonym* (*e.g.,* read and write) relations between [*actions*]. Furthermore, there are hierarchical conceptual relations (*i.e., isA*) between [*tasks*] as well, *e.g.,* <`Task(read, json file, in java)`, *isA*, `Task(read, json file)`>. A [*question*] describing a [*task*] with its elements (*i.e.,* [*action*], [*object*] and [*constraints*]) is an *instance of* the [*task*], *e.g.,* the question *"How to read a json file in java"* is an *instance of* `Task(read, json file, in java)`.

We can **annotate** all [*task*] elements in the [*question*] and obtain the **annotated** [*question*], *i.e., "How to read a json file in java"*. read, json file and in java are the annotated elements in the [*question*] with a role in the [*task*]. We use colors instead of tags to improve readability. Words in red, orange and blue represent annotated [*action*], annotated [*object*] and annotated [*constraint*], respectively. An annotated question may imply a linguistic pattern about describing a task and multiple questions may follow similar linguistic patterns, *e.g., "How to read a json file in java"* and *"How to read a pdf file in java"* only differ on the annotated [*object*].

3 MOTIVATIONAL STUDY

To understand issues with task-related questions, we sample a set of such questions from SO and investigate how they were changed. All the data used in this study is provided in the replication package [8].

Data. SO provides a data dump with 2,554,062 question title edit records for 16,663,358 questions [1]. One question could have multiple edit records. Among 16,663,358 questions, 13.4% questions have

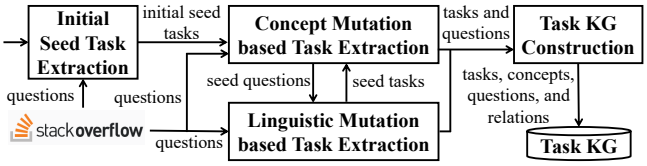

Figure 2: Task KG Mining of TaskKG4Q

at least one edit record. Of the 3,089,965 questions with the keyword *"how to"* in the title (representing typical how-to questions), 12.1% (373,933 of 3,089,965) have at least one edit record. Note that not all how-to questions have the *"how to"* keyword in the title. We randomly selected questions with title edit records from the dump and manually removed questions not related to how-to tasks. For example, the question *"React Native or React Js Memory Leak"* is about debugging, hence it is not a task-related question. The sampling and removal were conducted several times (100 questions per sample) until we obtained 410 task-related questions.

The sample size of 410 exceeds the sample size required to guarantee confidence interval of 5 at a 95% confidence level (*i.e.,* 384 [59]). The manual removal was conducted by two MS students independently (with more than four years of Java experience), with a Cohen's Kappa agreement [46] of 0.820, *i.e.,* almost perfect agreement. A third student was assigned to resolve the conflicts.

Protocol. We compared each title edit record with the previous version of the question title (*i.e.,* initial title or the last title edit record) and classified the edit record into four categories:

(1) Edit [*action*] (*e.g.,* change *"load a file"* to *"read a file"*);
(2) Edit [*object*] (*e.g.,* change *"load a file"* to *"load a large file"*);
(3) Edit [*constraint*] (*e.g.,* change *"load a file in java"* to *"load a file in java 8"*);
(4) Others (*e.g.,* adding *"how to"* to the question beginning without changing the task elements).

One edit record could be classified into multiple categories, as it may have multiple edits. This classification was done by two students (same students as above) independently. The Cohen's Kappa agreement [46] is 0.895 (*i.e.,* almost perfect agreement).

Results and Analysis. We identified 120 edit records in the Edit [*action*] category, 175 in Edit [*object*], 331 in Edit [*constraint*], and 150 in the Others category. 71.0% edit records are about modifying task elements. There are 307 edit records modifying only one [*task*] element; 55 edit records modifying two [*task*] elements; and 27 edit records modifying three [*task*] elements. On average, one question has 1.3 edit records (min. 1, med. 1, max. 5) and 1.1 edit records modifying [*task*] elements (min. 0, med. 1, max. 4). Overall, [*task*] elements from 92.7% of the questions have been edited.

Summary. Developers make frequent changes to SO task-related questions, suggesting that they are often incomplete or unclear.

4 TASK KG MINING

An overview of the task KG mining is presented in Fig. 2. It starts with **seed task extraction**, which identifies a small set of high-quality tasks as seed tasks based on manually defined linguistic patterns (Sec. 4.2). Based on the set of initial seed tasks, the approach employs an iterative process to **extract additional tasks and annotated questions**, in two steps: **concept mutation-based task extraction** (Sec. 4.3) and **linguistic mutation-based task extraction** (Sec. 4.4). The collection of produced tasks and annotated questions, with the relations between them constitute the task KG (Sec. 4.5). All these steps are automated except the definition of the linguistic patterns for the extraction of the initial seed tasks.

Concept mutation-based and linguistic mutation-based task extraction promote and complement each other. Concept mutation-based task extraction is to extract tasks with similar task elements and linguistic mutation-based task extraction is to extract tasks described with a similar linguistic pattern. During iteration, concept mutation-based task extraction can discover annotated questions with new linguistic patterns, and linguistic mutation-based task extraction can discover tasks with new concepts. Note that we do not aim to create new questions that do not exist in SO using mutation but use SO questions to validate mutated tasks and questions. Only valid tasks and questions are added to the task KG.

We compute a **confidence score** for each generated task and corresponding annotated questions. Tasks with high confidence scores are selected as seeds for the next concept mutation-based task extraction, while annotated questions with high confidence scores are selected as seeds for the linguistic mutation-based task extraction. When no new tasks can be produced, the approach ends with the resulting task KG as the output.

4.1 Running Example

Fig. 3 shows part of the task KG describing tasks related to reading files. The white ellipses denote tasks and the gray ellipses denote questions instance of tasks. The solid lines between the ellipses denote "*isA*" relations between tasks, and the dashed lines denote "*instanceOf*" relations between questions and tasks. For brevity, we omit in the figure some entities (*e.g.,* [*action*], [*object*], or [*constraint*]) and some relations (*e.g., hasAction*, *hasObject*). We construct the task KG to represent similar tasks to form a hierarchical knowledge structure, *e.g.,* read a pdf file with different languages or on different platforms, or read different files.

To mine such a task KG, we **obtain a list of questions from SO** as the corpus to run TaskKG4Q on, *e.g.,* questions with popular tags, such as, *"java"*, *"android"*, *"javascript"*, and *"python"*.

Before the iterative task and question extraction step, we select high-quality tasks from the corpus as **initial seeds**, based on manually defined linguistic patterns (see Sec. 4.2). For example, we extract `Task(read, pdf file, in java)` from the question *"how to read a pdf file in java"* and `Task(read, xml file, in java)` from the question *"how to read xml file in java"*, as initial seeds. In each round of the iteration, we first perform concept mutation-based task extraction and then linguistic mutation-based task extraction. Next, we illustrate how we perform a round of iteration starting from the seeds `Task(read, pdf file, in java)` and `Task(read, xml file, in java)` as shown in Fig. 4.

For **concept mutation-based task extraction**, first we obtain mutation tasks from the seed task based on domain knowledge (see Sec. 4.3.1). Those mutation tasks are related to the seed tasks at concept level and may be task candidates, *e.g.,* `Task(read, pdf file, in python)`, `Task(read, xml, in java)`. Then, we use those task candidates (including seed tasks and mutation tasks) to match with questions in the corpus to find the tasks' instance questions

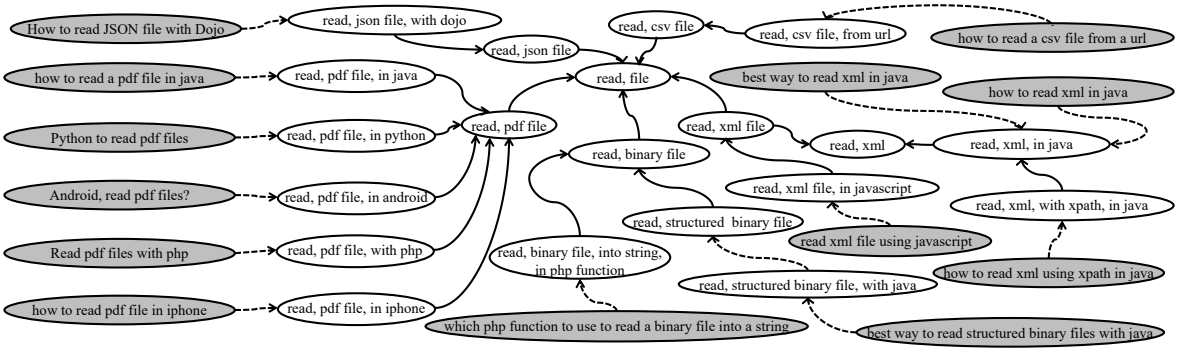

Figure 3: A part of the task KG related to reading files. White nodes are [*tasks*] and gray nodes are [*questions*]. Solid lines are *isA* relations and dashed lines are *instanceOf* relations.

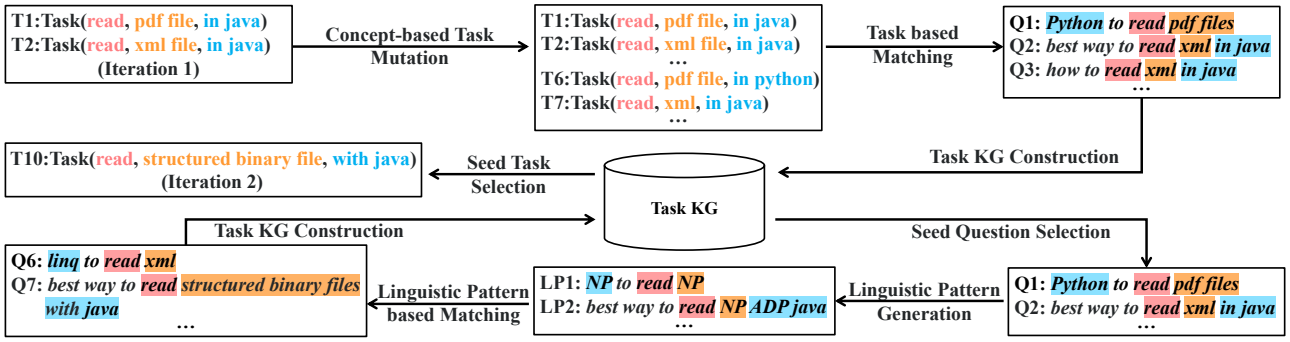

Figure 4: An Example of One Round Iteration for Task KG Mining

(see Sec. 4.3.2). For example, we can identify the question *"Python to read pdf files"* as an instance of `Task(read, pdf file, in python)`; and *"best way to read xml in java"* as an instance of `Task(read, xml, in java)`. We further annotate the elements of tasks in the matching questions to get the annotated questions, *i.e.,* *"Python to read pdf files"* and *"best way to read xml in java"*.

At the end of the concept mutation-based task extraction, tasks with corresponding annotated questions are added to the task KG and conceptual relations are also added during this process (see Sec. 4.5), *e.g.,* `<Task(read, pdf file, in python)`, *isA*, `Task(read, pdf file)>`. We further compute confidence scores for tasks and annotated questions in the task KG and select the Top-K unselected questions with the highest scores as the input for linguistic mutation based task extraction (see Sec. 4.3.4). As a result, we select *"Python to read pdf files"* and *"best way to read xml in java"* as the seeds, which describe tasks in linguistic patterns different from patterns defined for seed task extraction.

For **linguistic mutation based extraction**, we generate linguistic patterns from selected annotated questions by mutating the annotated elements (see Sec. 4.4.1), *e.g.,* linguistic patterns LP1: *NP to read NP* and LP2: *best way to read NP ADP java*. Based on the generated linguistic patterns, we can extract instance questions of the new task from the corpus (see Sec. 4.4.2), *e.g.,* the question *"linq to read xml"* matches with the first linguistic patterns and the annotated question *"linq to read xml"* is extracted; the question *"best way to read structured binary files with java"* matches with the second linguistic patterns and the annotated question *"best way to read structured binary files with java"* is extracted. Two new tasks `Task(read, xml, in linq)` and `Task(read, structured binary file, with java)` are extracted.

As before, the tasks with their corresponding annotated questions are added to the task KG and the necessary relations are added. At the end of the linguistic mutation based task extraction, we compute confidence scores for tasks and annotated questions in the task KG as well and select the Top-N unselected tasks with the highest scores as the seed tasks for the next concept mutation based extraction (see Sec. 4.4.3). For example, `Task(read, structured binary file, with java)` is selected as a seed task for the next iteration. In this way, we may extract new tasks related to binary files in the next concept mutation based task extraction.

4.2 Initial Seed Task Extraction

To obtain the initial seed tasks for the iterative extraction, we **use manually defined linguistic patterns** to extract high-quality tasks from given SO questions.

We inspected the 100 top voted questions with the *"java"* tag on SO, and then summarized the following linguistic patterns to identify and extract tasks.

- *How to V NP in NP*
- *How can I V NP in NP*
- *How do I V NP in NP*

Table 1: Mutation Operations With Examples

Mutation Operation	Mutated Elements	Task	Mutated Task
Action based	action	Task(read, pdf file, in java)	Task(load, pdf file, in java), Task(write, pdf file, in java)
Constraint Deletion based	constraint	Task(read, json file, in java, from server)	Task(read, json file, from server), Task(read, json file)
WikiData based	object, constraint	Task(read, pdf file, in java)	Task(read, pdf file, in python), Task(read, pdf file, in python 3)
SO Tag based	object, constraint	Task(read, xml file, in java)	Task(read, xml file, in jdk), Task(read, pdf file, in openjdk)
Morphology based	object, constraint	Task(read, pdf file, in java)	Task(read, pdf, in java), Task(read, file, in java)
Task KG based	object, constraint	Task(read, string, from file)	Task(read, json string, from file)

Where, *V* matches with a verb as the [*action*]; *NP* matches with a noun phrase as the [*object*]; *in NP* matches with a noun phrase starting with *"in"* as the [*constraint*]. The other parts of the pattern must be matched literally. In order to ensure the quality of the seeds, we limit the length of the matched noun phrases to four words or less (a convention in NLP [27, 32]). For example, the question *"How to read a pdf file in java"* will be matched with the first linguistic pattern and we extract `Task(read, pdf file, in java)`. The articles *"a"*, *"an"*, and *"the"* at the beginning of noun phrases are removed. We use spaCy [9] to implement pattern matching and task extraction.

Further, we **calculate the confidence score** for each extracted task. The confidence score of a task is the number of questions which contain the task's elements (*i.e.,* [*action*], [*object*], [*constraint*]). For example, assuming that *"read"*, *"pdf file"* and *"java"* appear in 100, 20, 200 questions respectively, then the confidence score for the `Task(read, pdf file, in java)` is 320 (*i.e.,* =100+20+200). Note that when calculating the number of occurrences of the constraints, we ignore the preposition, because the preposition may be different in different questions for the same constraint (*e.g., "in java"* and *"with java"*). Finally, we rank the extracted tasks from high to low according to their confidence scores, and then select the highest top-K tasks as initial seeds.

4.3 Concept Mutation based Task Extraction

To extract tasks related to seeds at concept level, we first identify new candidate tasks by mutating the task elements of given seed tasks based on domain knowledge. Then we verify the candidate tasks by matching them with SO questions and extract valid new tasks with corresponding annotated questions.

4.3.1 Concept-based Task Mutation. For any task discussed on SO, there may be related questions. For example, if a developer found a question about `Task(read, pdf file, in java)`, it is natural to think that there may be questions for similar tasks on SO, such as `Task(read, pdf file, in python)` and `Task(write, pdf file, in java)`. This association is based on the background knowledge of the developer, such as knowing that both Java and Python are programming languages and that read and write are opposite actions in programming. Based on this idea, we design different ways to obtain candidate tasks by mutating given tasks. Table 1 shows mutation operations with examples. Note that one mutation type may generate multiple candidate tasks for a given task.

Action based Mutation. Replace the [*action*] in the [*task*] with its synonyms and antonyms, *e.g.,* mutate the [*action*] from *"read"* to *"load"* or *"write"*. Xie *et al.* [70] summarized 87 common categories for describing API functionality, where each category contains verbs (provided by previous research [2]). For example, *"read"* and *"load"* belong to the same functionality category about reading something from sources. In this work, we treat the verbs in the same functionality category as synonyms. Among the 87 functionality categories, some represent opposite functionalities, *e.g.,* the categories represented by *"read"* and *"write"*. Functionality verbs from opposite categories are considered to be antonyms.

Constraint Deletion based Mutation. If the [*task*] has multiple [*constraints*], new tasks with all [*constraint*] subsets will be generated. For example, for `Task(read, json file, in java, from server)` shown in Table 1 with two constraints, three tasks are obtained after mutation (two tasks have one constraint and one has no constraint).

WikiData based Mutation. Replace concepts and [*constraints*] with similar concepts from WikiData. Liu *et al.* [39] identified 78,182 software concepts from WikiData. Many WikiData concepts have aliases, *e.g., "python 2"* and *"python 3"* are aliases for *"python"* [7]. If a concept in the [*object*] or [*constraints*] matches with any alias of a software concept, we replace it with: (1) other aliases of the matched software concept; and (2) aliases of sibling software concepts of the matched software concept. Two concepts are sibling concepts if they have the same hyponymy relations (*i.e., subclass of* [10], *instance of* [3], or *part of* [5]) with the same concept, *e.g.,* Java [4] and Python [7] are sibling concepts because they have an *instance of* relation to *"programming language"* [6].

SO Tag based Mutation. Replace [*object*] and [*constraint*] with similar concepts from SO tags. Each SO tag has synonyms, *e.g.,* the *"java"* tag has the synonyms *"jdk"*, *"jre"*, *"oraclejdk"*, and *"openjdk"*. If a concept in the [*object*] or the [*constraint*] matches with a synonym of a SO tag, we replace it with other synonyms of the same tags, *e.g.,* we replace *"java"* with *"jdk"*. Further, Zhang *et al.* [73] classified SO tags into 20 categories (*e.g.,* library, framework, tool). If a concept in the [*object*] or the [*constraint*] matches with a synonym of a SO tag, we replace it with other tags in the same category, *e.g.,* we replace *"gson"* with *"fastjson"*.

Morphology based Mutation. Replace noun phrases in [*object*] or [*constraint*] with all noun phrases extracted, *e.g.,* from the object *"pdf file"*, we extract two noun phrases *"pdf"* and *"file"* and replace *"pdf file"* with the two extracted noun phrases.

Task KG based Mutation. Replace the concept in the [*object*] or [*constraint*] with other sub-concepts in our task KG, *e.g., "string"* could be mutated to *"json string"* because we extracted two objects *"json string"* and *"string"* from tasks in previous iterations (*e.g.,* `Task(parse, json string, in python)` and `Task(read, string, from console)`) and create a relation <json string, *isA*, string> in the task KG (see Sec. 4.5). In this way, we can mutate the concepts based on the previous task extraction iterations.

4.3.2 Task based Matching. We match SO questions with the candidate tasks (seed tasks and mutated tasks). If a question includes the [*action*], [*object*] and [*constraint*] of one task at the same time, we consider this question to match with the task and it is an instance of the task. Note that a question includes a [*constraint*] if it includes

the noun phrase of the [*constraint*]. We ignore the preposition in the [*constraint*] because the same [*constraint*] may have different prepositions or no prepositions at all, *e.g., "Python to read pdf files"*. The matching is lemmatization-based rather than literal-based, thus *"reading"* and *"pdf files"* in a question may match with *"read"* and *"pdf file"* in a task respectively. In this way, we verify the mutated tasks with real questions. Invalid mutation tasks that have no corresponding question are filtered out. Sentence parsing errors caused by NLP tools will not affect this step, because the matching is based on task elements and not on the sentence parsing result.

4.3.3 Task and Annotated Question Completion. For each matching question with a task, we further annotate the task components (*i.e.,* [*action*], [*object*], [*constraint*]) in the question to obtain the annotated question for the task. After that, we analyze the entire question and try to complete and correct the annotated question and the task. We attempt to extract more prepositional phrases as additional constraints, and identify whether the sentence has a more suitable direct object as the [*object*]. For example, based on `Task(read, file, in java)`, we identify the question *"how to read data from file in java with stream"* as a matching question. The annotated question will be *" how to read data from file in java with stream"* at first. After completion, the annotated question will be *"how to read data from file in java with stream"* corresponding to a new task `Task(read, data, from file, in java, with stream)`. In some cases, the preposition in a constraint could be missing, *e.g.,* *"Android, read pdf files"*. If the concept in the constraint already exists in the task KG, we add the preposition that is most commonly used with the concept as a supplement. We add a default preposition *"in"* for the constraint, *e.g.,* the corresponding task after completion is `Task(read, pdf file, in android)` for the annotated question with a missing preposition.

4.3.4 Seed Question Selection. To select high-quality annotated questions as seeds for the linguistic mutation based task extraction (Sec. 4.4), we compute the confidence scores for all tasks and annotated questions in the task KG and select seed questions according to the confidence scores.

We define the confidence score for questions differently than for tasks before (which is a frequency count). We consider the *coverage* and *selectivity* of a task corresponding to the question, defined in Eq. 2 and Eq. 5 (Eq. is short for Equation), respectively. *Task coverage* reflects the task's availability in the corpus, and it is based on the number of questions related to the task in the corpus. If a task covers more questions in the corpus, it is more likely to lead to the extraction of new tasks. *Task selectivity* reflects the task's ability to select questions from the corpus, and it is based on the number of extracted annotated questions related to the task. If there are more task-related questions extracted, then the task is more likely to help extract new tasks. Selectivity is computed using the annotated questions that have been extracted, while coverage uses all the questions in the corpus.

We introduce notations necessary for formally defining the confidence score. An annotated question is marked as aq; a task is marked as t; the task for which aq is an instance is marked as t_{aq}. e represents an element of a task t, *i.e.,* [*action*], [*object*], or [*constraint*], and w is a word in a task t (except for prepositions in constraints). EN_t and WN_t represent the number of elements or words in a task t, respectively.

The confidence score of a task t is defined using Eq. 1.

$$Score(t) = 1 - (1 - Coverage(t)) * (1 - Selectivity(t)) \quad (1)$$

We use the Noisy-OR model [61] to combine the coverage and selectivity. In this way, if either coverage or selectivity is large, then the final confidence will be large.

We measure task coverage from two perspectives: (1) task element level and (2) word level, using Eq. 3 and Eq. 4, respectively. Task coverage is then the average of task element level coverage $Coverage_e(t)$ and word level coverage $Coverage_w(t)$.

$$Coverage(t) = (Coverage_e(t) + Coverage_w(t))/2 \quad (2)$$

Consider QN is the number of questions in the corpus; QN_e is the number of questions containing the element e of a task t; and QN_w is the number of questions containing word w of a task t.

When computing $Coverage_e(t)$, we first compute the coverage of each element e from task t in the corpus, that is, QN_e/QN. Because the number of questions in the corpus may be large, we use the natural logarithm of both QN_e and QN to reduce the gap, *i.e.,* $lnQN_e/lnQN$. The product of the maximum and average of the coverage of each element $lnQN_e/lnQN$ is taken as $Coverage_e(t)$, ranging from 0-1. $Coverage_w(t)$ is defined in the same way, except that the proportion of each word w is calculated as $lnQN_w/lnQN$.

$$Coverage_e(t) = \max_{e \in t}(lnQN_e/lnQN) * \frac{\sum_{e \in t} lnQN_e/lnQN}{EN_t} \quad (3)$$

$$Coverage_w(t) = \max_{w \in t}(lnQN_w/lnQN) * \frac{\sum_{w \in t} lnQN_w/lnQN}{WN_t} \quad (4)$$

Only if the maximum coverage and average coverage of all elements are high, the element level coverage will be high.

We measure the selectivity from two perspectives: (1) task element level and (2) word level, using Eq. 6 and Eq. 7 respectively. Task selectivity is the average of task element level selectivity $Selectivity_e(t)$ and word level selectivity $Selectivity_w(t)$.

$$Selectivity(t) = (Selectivity_e(t) + Selectivity_w(t))/2 \quad (5)$$

In Eq. 6 and Eq. 7, AQN is the number of annotated questions in the task KG; AQN_e is the number of annotated questions containing the element e of a task t; and AQN_w is the number of annotated questions containing word w of a task t. $Coverage_w(t)$ is defined in the same way, except that the proportion of each word w is computed as $lnQN_w/lnQN$.

$$Selectivity_e(t) = \max_{e \in t}(lnAQN_e/lnAQN) * \frac{\sum_{e \in t} lnAQN_e/lnAQN}{EN_t} \quad (6)$$

$$Selectivity_w(t) = \max_{w \in t}(lnAQN_w/lnAQN) * \frac{\sum_{w \in t} lnAQN_w/lnAQN}{WN_t} \quad (7)$$

Finally, the confidence score of an annotated question aq is defined using Eq. 8. We consider two factors for computing the confidence score of an annotated question: (1) task confidence and (2) task information ratio. In Eq. 8, harmonic mean is used to combine task confidence and task information ratio, as we deem both equally important quality factors.

$$Score(aq) = \frac{2 * Score(t_{aq}) * WN_t/WN_{aq}}{Score(t_{aq}) + WN_t/WN_{aq}} \quad (8)$$

The higher the confidence of the corresponding task t_{aq} is, the higher the quality of the annotated question t_{aq}. As for the task information ratio, we consider it as the ratio of the number of words in task WN_t and the number of words in question WN_{aq}. The larger the ratio, the more standard and concise is the way in which the question describes the task, and the more likely the linguistic pattern generated from this annotated question could match with new questions. For example, for the same task `Task(read, json`

`file, in java`), *"read a json file in java"* is better than *"read a json file that is very large in java"* because the previous question contains more words related to the task and is more likely to have other questions sharing a similar linguistic pattern.

Note that we only used element-level coverage and selectivity in the beginning. We found that the seed tasks selected are overlapping on high-frequency task elements. Therefore, to increase the variability of seeds, we added word-level coverage and selectivity. In this way, we can select seed tasks with low-frequency task elements yet with shared words with high-frequency task elements, *e.g.,* the low-frequency task element "image file" benefits from the high-frequency task element "file".

When new annotated questions and tasks are added to the task KG, we re-compute the confidence scores for all tasks and annotated questions in the task KG. The seeds selected for the next iteration may be affected by the scores. For example, if in one iteration, we extract many tasks with annotated questions relevant to *"pdf file"*, this may indicate that there are many questions discussing tasks related to pdf files in the corpus. The confidence scores for previously extracted tasks and annotated questions related to *"pdf file"* will become higher and those tasks and annotated questions are more likely chosen as seeds for the next task extraction.

4.4 Linguistic Mutation based Task Extraction

Developers usually follow certain linguistic patterns to describe tasks, *e.g.,* the linguistic patterns that we used in the seed task extraction (see Sec. 4.2). To extract tasks with similar linguistic patterns to the ones contained in the seed tasks, we first mutate the annotated elements of selected seed questions to generate linguistic patterns automatically. Based on the generated linguistic patterns, we identify new annotated questions and extract new tasks from these new annotated questions.

4.4.1 Linguistic Pattern Generation. For a given annotated question with N annotated elements, we mutate it to generate multiple linguistic patterns. We randomly select 1, 2, 3, ..., N-1 annotated elements in turn for mutation. Different annotated elements are mutated to different linguistic elements:

(1) [*action*] is mutated to *V ADP?*, matching with any verb or verb phrase (*e.g.,* set up), where ADP refers to a preposition;

(2) [*object*] is mutated to *NP*, matching with any noun phrase;

(3) [*constraint*] is mutated to *ADP NP*, matching with any noun phrase starting with a preposition.

In order to ensure the quality of the generated linguistic patterns, we make sure that at least one annotated element of the generated pattern is not mutated. All other words in the annotated question except for mutated elements remain in the generated linguistic patterns. For example, from the annotated question, *"How to read pdf file in java"*, we generate six linguistic patterns (*i.e.,* we mutate [*action*], [*object*], [*constraint*], [*action*] + [*object*], [*action*] + [*constraint*], [*object*] + [*constraint*]). *How to read NP ADP NP* is one of the linguistic patterns generated by mutating one object and one constraint. In order to generalize the generated patterns and be able to match them with questions with subtle differences, we allow for missing articles, numbers, and punctuation from the original annotated question. We mark these as <DET?>, <NUM?> and <PUNCT?>, respectively in the pattern.

4.4.2 Linguistic Pattern based Matching. We use the generated linguistic patterns to find matching SO questions and extract the corresponding annotated questions and tasks, following the same process as described in Sec. 4.2. After extracting tasks with annotated questions, we follow the same steps as described in Sec. 4.3.3 to complete the annotated questions and the tasks. We perform lemmatization of the actions, objects and constraints, and remove stop words (*e.g.,* a, an, my) at the beginning of the involved noun phrases. In order to extract more possible questions without introducing too much noise, we remove annotated questions where noun phrases in the object or constraints have more than six words (based on our experience), which is more relaxed than the threshold in Sec. 4.2.

4.4.3 Seed Task Selection. To select high-quality tasks as seeds for concept mutation based task extraction (Sec. 4.3), we compute the confidence scores for all tasks and annotated questions in the task KG and select seeds according to the confidence scores. The confidence score is the same as for the seed question selection in Sec. 4.3.4.

4.5 Task KG Construction

To build the hierarchical conceptual structure in the task KG, we identify relations between concepts and tasks, when extracted tasks and annotated questions are added to the task KG, at the end of the task extraction steps. When adding tasks to the task KG, we add the task elements (*i.e.,* [*actions*], [*objects*], [*constraints*]) and build the *has*-relations (*i.e., hasAction*, *hasObject*, *hasConstraint*), while *instanceOf* relations are added between the annotated questions and their corresponding tasks.

Further, we identify the relations between concepts and add them to the task KG. This part is very similar to the previous concept based mutation described in Sec. 4.3.1. The *synonym* and *antonym* relations between *actions* are identified based on the functionality categories provided by Xie *et al.* [70]. We identify *isA* relations between concepts C_1 and C_2 referenced by [*objects*] or [*constraints*] in three ways:

(1) WikiData hyponymy relations. C_1 and C_2 have corresponding software concepts SC_1 and SC_2 respectively, and SC_1 and SC_2 have one of three hyponymy relations (*i.e.,* subclass of, instance of, part of) between them, *e.g., "json"* and *"file format"*.

(2) SO tag categorization. If C_2 is matching with one of the 20 categories defined by Zhang *et al.* [73] and C_1 is matching with one of the tags belonging to the category.

(3) Morphology characteristics. If C_2 is a prefix or suffix of another concept C_1.

As for tasks, a task has an *isA* relation to another task if it contains additional constraints (*e.g.,* `Task(read, pdf file, in java)` and `Task(read, pdf file)`) or it refines the object or constraint with more specific sub-concepts (*e.g.,* `Task(read, large text file)` and `Task(read, text file)`). If a task with fewer constraints does not exist in the task KG, the task is added to the task KG.

4.6 Resulting Task KG

We could mine the task KG from all SO questions. However, in order to improve efficiency, we only selected questions tagged with at least one of the six most popular tags (*i.e.,* java, python,

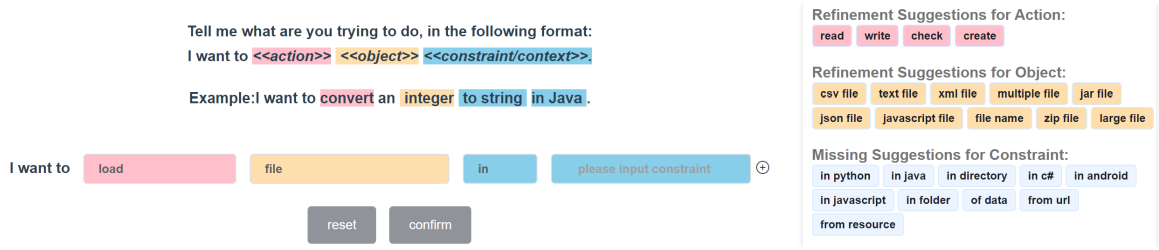

Figure 5: User Interface for TaskKG4Q

javascript, php, c#, android) from the SO data dump [1], with at least one upvote, as the corpus. Note that our approach is not limited to specific programming languages. We only consider the title of questions. The corpus includes 304,976 questions. We further filtered out low-quality questions or questions not related to tasks if one of the following criteria is met:

(1) questions containing no verbs or only passive verbs;

(2) questions with more than 15 words (5% of the questions were filtered out based on this threshold);

(3) questions starting with *"why"*, *"what does it mean"* or *"when"* or including the words *"difference"*, *"vs"*, *"exception"* or *"error"*. These keywords indicate that the question is related to technical comparison or debugging, not to programming tasks.

After filtering, we obtained 257,430 questions.

For seed task extraction, we extracted 934 tasks, using the three manually defined linguistic patterns mentioned before, and selected the top 500 tasks with the highest confidence score as the initial seed tasks for TaskKG4Q. In each iteration, we selected the top 500 annotated questions as the seeds for linguistic mutation based task extraction and selected the top 200 tasks as the seeds for concept mutation based task extraction. The selected confidence thresholds are both 0.5. The thresholds were determined via trial-and-error.

The resulting task KG contains 266,659 *tasks* and 102,283 *annotated questions*. The task KG has 1,809 *actions*, 58,825 *objects*, 124,891 *constraints* and 97,875 *concepts*. There are 40,380 *isA* relations between tasks and 292,114 *isA* relations between concepts.

5 A TOOL FOR INTERACTIVE HOW-TO QUESTION FORMULATION

We developed an interactive tool to help developers write how-to questions based on TaskKG4Q. Fig. 5 shows the user interface of our tool. The tool enforces the canonical format of <*action*, *objects*, and *constraints*> for the how-to question. The users can use the drop-down boxes to fill in task elements (*i.e.,* the [*action*], [*object*] and [*constraints*]) highlighted in different colors, click on the suggestions, or type freely. If the users need to write more than one constraint in the task, they can click the add button on the right to add more input boxes about constraints.

As the users fill in the task elements, the tool will provide on-the-fly suggestions on the right based on what they already wrote. The suggested options in the drop-down boxes are updated as well, accordingly. The tool shows two types of suggestions: missing suggestions and refinement suggestions. For example, if no [*action*] is entered, the right side will show missing suggestions for the [*action*]. If the [*action*] is entered and no [*object*] is entered, then the right side will show missing suggestions for the [*object*], and so on. The missing suggestions are based on the popularity of candidate suggestions. We define the popularity of a task element as the number of questions that includes it. If the user has entered some task elements, such as, [*action*] and [*object*], then the tool selects questions that contain at least one of the entered task elements as candidate questions from the task KG. We determine the popularity for all candidate suggestions using Eq. 9 and list the top-10 as missing suggestions based on the popularity. The users can expand the suggestion list to see more suggestions, if needed.

In Eq. 9, e_c is a candidate suggestion task element; E is a set of task elements have been entered and e_e is a task element that has been entered. $CoOccur(e_c, e_e)$ is the number of questions including e_c and e_e at the same time and $Occur(e_c)$ is the number of questions including e_c.

$$Popularity(e_c) = \begin{cases} \sum_{e_e \in E} CoOccur(e_c, e_e) & E \neq \emptyset \\ Occur(e_c) & E = \emptyset \end{cases} \tag{9}$$

We remove the concepts that are sub-concepts or siblings of the entered task elements from the suggestions. For example, if *"in java"* is entered as the [*constraint*] then *"python"* and *"java 8"* will be removed from the suggestions for the missing [*constraint*].

We also show refinement suggestions for the task elements already entered. For the [*object*] and [*constraint*], the refinement suggestions are the sub-concepts in the task KG, ranked by popularity. For the [*action*], the refinement suggestions are [*actions*] with higher popularity than the current [*action*]. For example, if the current [*action*] and [*object*] are *"load"* and *"file"*, the tool will show *"read"* as a refinement suggestion for the [*action*] because we find that more questions are an instance of `Task(read, file)` than `Task(load, file)`.

6 EVALUATION

We conducted empirical studies to evaluate the intrinsic quality of the task KG and TaskKG4Q's effectiveness in helping developers formulate specific how-to questions. As extrinsic evaluation, we investigated TaskKG4Q's usefulness in helping developers complete actual programming tasks. More specifically, we focus on answering the following research questions:

RQ1: What is the intrinsic quality of the task KG?

RQ2: How effective is TaskKG4Q in helping developers formulate specific how-to questions?

RQ3: How useful is TaskKG4Q in helping developers complete programming tasks?

All the data used in these studies is provided in the replication package [8].

6.1 RQ1 Intrinsic Quality

We evaluated the intrinsic quality of the task KG by assessing the correctness of the annotated questions in the task KG.

6.1.1 Protocol. Similar to previous studies [33, 40, 67], we adopt a sampling method [59] to ensure that ratios observed in the sample generalize to the population within a certain confidence interval at a certain confidence level. For a confidence interval of 5 at a 95% confidence level, the required sample size is 384. We randomly selected 384 annotated questions with corresponding tasks from the task KG. We invited two Master students (not affiliated with this work) with extensive development experience and familiarity with SO to assess the label accuracy of the annotated questions independently. The criterion is that the question is an instance of the corresponding task and that all elements are annotated correctly in the question. For each sampled question, if it was assessed differently by the two students, a third student was assigned to give an additional assessment to resolve the conflict by a majority-win strategy.

6.1.2 Results. The accuracy as determined by the annotation is 75.0%, *i.e.,* 288 annotated questions are correct and 96 annotated questions are at least partially incorrect (*e.g.,* one task element annotated incorrectly). The Cohen's Kappa agreement is 0.885, *i.e.,* almost perfect agreement. We analyzed reasons for the errors. Some errors are caused by the NLP tool. For example, we only extracted `(disable on, google map)` from the question "disable double left click on google map" and failed to identify the correct object. The NLP tool identifies the action "disable" as an adjective and can not identify "double left click" as the object of "disable". Another reason is that questions are not related to a task, *e.g.,* we extracted `Task(enter, custom text)` from the question "combobox doesn't allow enter custom text if databinding is used". However, in this case, we argue that the task is still somewhat meaningful. Future work will improve the accuracy further, for example, by training a question classifier to identify task-related questions similar to the work of Beyer *et al.* [17].

6.1.3 Summary. 75.0% of the annotated questions with tasks extracted by TaskKG4Q from the question corpus are correct.

6.2 RQ2 Effectiveness

We evaluated the effectiveness of TaskKG4Q by asking participants to formulate how-to questions for programming tasks and comparing with two baselines.

6.2.1 Programming Tasks. We randomly selected 18 questions from the task-related questions we collected in the motivational study from Sec. 3 as programming tasks. For each selected programming task, we removed the original title and only kept the question body and the question tags as the context for formulating how-to questions. Those programming tasks cover different programming languages (*e.g.,* Java, JavaScript) and different topics (*e.g.,* database, user interface).

6.2.2 Baselines. We compared TaskKG4Q with two baselines: Chatbot4QR [73] and WithoutTool (*i.e.,* participants formulate how-to question without any tool). Chatbot4QR is an interactive query refinement approach designed by Zhang *et al.* for question retrieval. Given a query, Chatbot4QR can generate several clarification questions to interact with the user, *e.g., "What programming languages do you prefer? e.g., java or c#".* Chatbot4QR combines the answers of clarification questions with the original query to generate a refined query and retrieve SO questions based on the refined query. TaskKG4Q is different from Chatbot4QR because we help developers construct the question from zero while Chatbot4QR helps developers refine an existing question. To enable the comparison, we asked the participants to formulate an initial how-to question without any tool first and then refining the initial question with Chatbot4QR. We used the implementation of Chatbot4QR provided by Zhang *et al.* in their replication package.

6.2.3 Protocol. We invited 9 students (not affiliated with this work) with 1-5 years of development experience to conduct this experiment. We first randomly divided the 18 tasks into 6 groups of 3 tasks each. The tasks in the same group are assigned to the same three participants to formulate how-to questions, with different approaches. We use a within-subject design where each participant only uses one approach per task, *i.e.,* participant X formulates a question for task A without any tool, for task B with Chatbot4QR, and for task C with TaskKG4Q; participant Y formulates a question for task A with Chatbot4QR, for task B with TaskKG4Q, and for task C without any tool; and participant Z formulates a question for task A with TaskKG4Q, for task B without any tool, and for task C with Chatbot4QR. As a result, each participant formulates how-to questions for tasks from two groups (*i.e.,* six tasks): two task with TaskKG4Q, two tasks with Chatbot4QR, and two tasks without any tool. When participants formulate how-to questions for tasks, the body and tags of the original questions are provided as the context. We collected 54 how-to questions formulated by the participants for 18 tasks, three questions per task formulated with three approaches.

We asked another two MS students to rank the how-to questions for each task according to the quality of the questions. The quality of a question was evaluated by its completeness, understandability, and conciseness. The criterion for completeness is whether the question contains all the necessary and specific information for the task to be understood and answered correctly. Understandability and conciseness require that the question is understandable and contains no (or very little) unnecessary or redundant information, *e.g.,* without complicated clauses. The evaluators were asked to consider all criteria together, when ranking the questions for a task. Note that they were not informed how the questions were generated or the purpose of the study. Moreover, the three questions for each task were presented in random order for them to rank. If the rankings of the three questions for one task are different, a third MS student is assigned to resolve the conflict by a majority-win strategy. The Cohen's Kappa agreement is 0.703, representing substantial agreement.

6.2.4 Results. The results of the comparison are shown in Fig. 6. The average rankings of TaskKG4Q, Chatbot4QR, and WithoutTool are 1.4, 2.3, and 2.3 respectively, *i.e.,* participants were able to formulate better how-to questions with the help of TaskKG4Q. TaskKG4Q ranks higher than Chatbot4QR for 77.8% of the questions; TaskKG4Q ranks higher than WithoutTool for 72.2% of the questions; and Chatbot4QR ranks higher than WithoutTool for

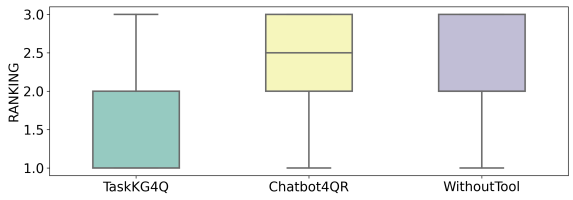

Figure 6: Ranking of the how-to questions formulated with TaskKG4Q and the baselines

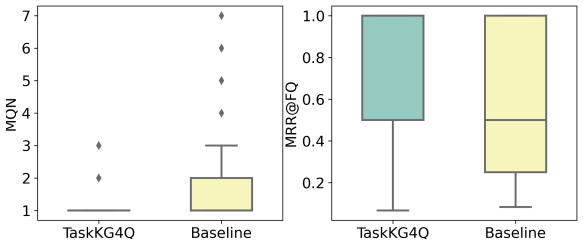

Figure 7: Comparison between TaskKG4Q and Baseline

50.0% of the questions. We used Welch's t-test [69] to verify the statistical significance of the difference between TaskKG4Q, Chatbot4QR, and WithoutTool on rankings. The differences between TaskKG4Q and WithoutTool as well as TaskKG4Q and Chatbot4QR are statistically significant ($p << 0.05$ for both).

6.2.5 Summary. TaskKG4Q helps developers formulate better how-to questions than Chatbot4QR or without tool support.

6.3 RQ3 Usefulness

We evaluated the usefulness of TaskKG4Q by asking participants to complete a set of programming tasks. When they need to learn how to solve a specific task, we asked them to find the specifics on SO. They did that by writing questions on their own and with the TaskKG4Q tool. We then compared the questions written with or without the tool support and assess which ones are better.

6.3.1 Participants. Novices are the main audience of TaskKG4Q, as we expect experienced developers can write good questions without tool support. Hence, we focused our recruitment on novices in a certain programming language (*i.e.,* familiar with the basic syntax, but not familiar with many APIs). Such novices will have at least one question and seek answers, when completing their tasks. To recruit participants, we used a questionnaire to select 16 qualified students from a class of 49 students. The questionnaire assessed their programming knowledge and whether they would need help to complete the four given programming tasks.

6.3.2 Programming Tasks. We designed four small programming tasks such that they contain several features and that participants may need to search on SO to find the solution (*e.g.,* save a json file in Java). These features are independent, meaning that participants should not find the answer to one, while looking for another. The programming tasks are extracted from lab assignments from a Data Structures class and modified slightly to include the above mentioned features. The four tasks are:

T1: *import a list of students with their grades in json format and convert it to a csv file, which also includes their GPA.*

T2: *get all file names in a given directory, rank them by size, and calculate the file numbers for each type of file extension.*

T3: *read a text file and determine the frequency of each word and output the top 100 most frequent words to a text file.*

T4: *read a list of students' names from the console, then randomly divide them into N groups, and save the groups into a json file.*

6.3.3 Protocol. We divided participants into two comparable groups PA and PB using similar programming languages, *i.e.,* in each group, there are 4, 3, and 1 participants using Python, Java, and C++ respectively, because they are novices in those languages. The questions were divided into two roughly equal groups TA (T1 + T2) and TB (T3 + T4), based on task difficulty. For PA, participants complete TA with the baseline (*i.e.,* without our tool) and TB with our tool. For PB, participants complete TB with the baseline and TA with our tool.

When completing a task, an example input and output are provided with more detailed task context (included in our replication package [8]). Participants must submit the complete code for each task and the code is reviewed by the authors to confirm its correctness. While the participants complete a task, they can search on SO and we ask them to record their queries. If they reformulate the questions, we record the reformulations too. If they found an answer they want using a question, we asked them to record the answer with the ranking of the answer in the search results list. When the participants use our tool, they will write the question and refine it with the help of the tool, then use it to search on SO. On the other hand, participants using the baseline can only construct the query by themselves and reformulate the query based on the feedback from SO search results. Each participant was first assigned tasks where they had to manually formulate the questions, and then tasks where they used the tool, to avoid a learning effect.

After completing all the programming tasks, we also conducted interviews to get participants' feedback on our tool. Participants were asked to evaluate our tool in terms of usefulness and usability on a 4-points Likert scale (1-disagree; 2-somewhat disagree; 3-somewhat agree; 4-agree) by rating the following statements:

Usefulness: The tool's suggestions were helpful for writing the questions.

Usability: The tool was easy to use.

6.3.4 Results. All participants agreed that the tool is helpful (43.8% somewhat agreed and 56.2% agreed) and it is easy to use (50.0% somewhat agreed and 50.0% agreed).

From the experiment, we collected 216 queries (80 with our tool and 136 with the baseline). Only 4 of these are formulated the same in the two conditions. We manually analyzed the queries and grouped them based on features that participants needed to complete, *i.e.,* we aligned queries by features. If queries related to one feature are only raised when using our tool or the baseline, we remove the queries to facilitate comparison. After filtering, we obtained 189 queries (70 with our tool and 119 with the baseline) for 12 features. We calculated MQN (mean query number) for our tool and the baseline, *i.e.,* for each feature, the number of queries that participants have to write to obtain the correct answers on average. We also computed MRR@FQ (mean reciprocal rank for the first

query) for our tool and the baseline. MRR (mean reciprocal rank) is the reciprocal rank of the correct answer in the search result list.

Fig. 7 shows MQN and MRR@FQ when using our tool and the baseline. Using our tool, participants found answers with fewer queries (1.3 vs 2.3 on average), compared to the baseline. At the same time, the reciprocal rank of the correct answer with the first query is better (0.72 vs 0.62 on average) compared to the baseline. We used Welch's T-test [69] for verifying the statistical significance of these differences. The difference in MQN is statistically significant, with $p = 0.0001 << 0.05$. The difference in MRR@FQ is not statistically significant at an alpha level of 0.05, with $p = 0.1072$.

In addition, the informal feedback we received from participants shows that when using the baseline they are often frustrated because the queries they write cannot match with questions on Stack Overflow, but the answers they actually want are often found by changing the query with appropriate technical terms. When using our tool, participants could write queries step by step guided by our tool and suggestions provided are useful for them to choose more appropriate technical terms. Several participants also asked for similar support tools for other types of questions, *e.g.,* questions related to bug fixes. We will consider this in future work.

6.3.5 Summary. TaskKG4Q can help novices write better how-to questions for SO and they consider it easy to use.

7 THREATS TO VALIDITY

A potential threat to internal validity stems from the use of the natural language processing library, spaCy. We extract seed tasks and perform linguistic mutation with spaCy. No natural language processing library achieves 100% accuracy on any large data set, and spaCy's performance was found to be on par with the state of the art [24] and outperforming other libraries when applied to software documentation [13]. We design our approach to tolerate errors of NLP tools to a certain extent as explained in Sec. 4.3.2 and 4.4.2. The empirical study and the evaluation share common threats to validity. A threat to the internal validity is the subjective judgments in different parts. To alleviate this threat we used at least two judges and reported the agreement for each subjective judgment or the corresponding statistical significance. A threat to the external validity is the limited number of subjects (*e.g.,* programming tasks, participants) considered in different parts. We also cannot claim generalizability of TaskKG4Q beyond questions related to the six most popular SO tags.

8 RELATED WORK

Previous research has categorized Stack Overflow questions [14, 18, 38, 47, 57, 64] and how-to questions are identified as an important question category on Stack Overflow. For example, Treude *et al.* [64] identified ten categories of Stack Overflow questions: *i.e., how-to*, *discrepancy*, *environment*, *error*, *decision help*, *conceptual*, *review*, *non-functional*, *novice*, and *noise*. In this paper, we focus on formulating how-to questions.

Existing studies focus on extracting task-related knowledge [62, 65] or similar verb phrases from identifier names [28, 29, 31, 58], API functionality description [70], documentation [65], and comments [58]. For example, Treude *et al.* [65] extracted tasks from documentation to help developers navigate documentation. Different from these works, we standardized the representation of tasks (*i.e.,* action, object, constraints), and then designed a top-down and bottom-up combination to iteratively extract tasks from a large-scale corpus with the hierarchical conceptual relations between tasks.

In the field of software engineering, some studies build other types of KGs from different sources, such as KGs for bugs [68], API caveats [33], domain terminology [67, 71], API concept and descriptive knowledge [39], and API comparison [40]. Our work is the first creating a task KG for how-to questions from Stack Overflow.

Many studies focus on automatically reformulating queries by expanding them with relevant terms [21–23, 26, 30, 35, 37, 42, 44, 45, 49, 52–54, 56, 60] for different retrieval tasks, *e.g.,* bug localization and code search. For example, Hill *et al.* [30] reformulate queries with natural language phrasal representations of method signatures. Similarly, Lu *et al.* [43] also extend queries with synonyms based on WordNet to improve the hit rate of code search. For searching on Stack Overflow, Zhang *et al.* [73] designed Chatbot4QR to help developers refine their queries, suggesting tags from existing posts. Similarly Cao *et al.* [20] proposed SEQUER to support automated software-specific query reformulation based on query logs provided by Stack Overflow. In contrast, we focus here on helping developers formulate questions from scratch, rather than refining an existing query, and we only focus on how-to questions.

Calefato *et al.* [19] provided suggestions for writing questions on Stack Overflow, while other research studied the factors that affect the quality of questions [16, 55] and how to automatically predict the quality of questions [15, 50]. Unlike that research, we help developers write quality how-to questions from the beginning.

9 CONCLUSIONS AND FUTURE WORK

We observed that Stack Overflow questions are subject to many edits, indicating that developers struggle to write good questions from scratch. We also found that 71% of the edits to how-to questions are modifying the elements of the underlying task (*i.e.,* [*actions*], [*objects*], [*constraints*]). This is evidence that developers could benefit from tool support in formulating how-to questions.

We posit that how-to programming questions posed by developers are important and frequent enough that they need specialized tool support, as opposed to *"one size fits all"* query reformulation approaches. Our solution (TaskKG4Q) is an interactive approach, that leverages the structure and content of past how-to questions.

We built a task Knowledge Graph from 257,430 SO questions, extracting 1,809 *actions*, 58,825 *objects*, 124,891 *constraints*, 97,875 *concepts*, and relations between them. We also found that using the task KG to guide novices when formulating how-to questions related to programming tasks helps them produce better questions.

In the future, we will combine TaskKG4Q with existing automatic Q&A systems (*e.g.,* AnswerBot [72] or APIBot [63]).

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China under Grant No. 61972098.

REFERENCES

[1] 2021. *Stack Overflow data dump version from March 4, 2021.* Retrieved September 4, 2021 from https://archive.org/download/stackexchange/

[2] 2022. *FuncVerbNet.* Retrieved March 4, 2022 from https://github.com/FudanSELab/funcverbnet

[3] 2022. *instance of.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Property:P31

[4] 2022. *Java.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Q251

[5] 2022. *part of.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Property:P361

[6] 2022. *Programming Language.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Q9143

[7] 2022. *Python.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Q28865

[8] 2022. *Replication Package.* Retrieved September 5, 2022 from https://fudanselab.github.io/Research-FSE2022-taskKG

[9] 2022. *SpaCy.* Retrieved March 4, 2022 from https://spacy.io

[10] 2022. *subclass of.* Retrieved March 4, 2022 from https://www.wikidata.org/wiki/Property:P279

[11] 2022. *WikiData.* Retrieved March 4, 2022 from https://wikidata.org/

[12] Ahmad Abdellatif, Khaled Badran, Diego Costa, and Emad Shihab. 2021. A Comparison of Natural Language Understanding Platforms for Chatbots in Software Engineering. *IEEE Transactions on Software Engineering* (2021). https://doi.org/10.1109/TSE.2021.3078384

[13] Fouad Nasser A Al Omran and Christoph Treude. 2017. Choosing an NLP Library for Analyzing Software Documentation: A Systematic Literature Review and a Series of Experiments. In *14th International Conference on Mining Software Repositories, MSR 2017, May 20-28, 2017, Buenos Aires, Argentina.* 187–197. https://doi.org/10.1109/MSR.2017.42

[14] Miltiadis Allamanis and Charles A. Sutton. 2013. Why, When, and What: Analyzing Stack Overflow Questions by Topic, Type, and Code. In *10th Working Conference on Mining Software Repositories, MSR 2013, May 18-19, 2013, San Francisco, CA, USA.* 53–56. https://doi.org/10.1109/MSR.2013.6624004

[15] Antoaneta Baltadzhieva and Grzegorz Chrupala. 2015. Predicting the Quality of Questions on Stackoverflow. In *Recent Advances in Natural Language Processing, RANLP 2015, 7-9 September, 2015, Hissar, Bulgaria.* RANLP 2015 Organising Committee / ACL, 32–40.

[16] Antoaneta Baltadzhieva and Grzegorz Chrupala. 2015. Question Quality in Community Question Answering Forums: a survey. *SIGKDD Explor.* 17, 1 (2015), 8–13. https://doi.org/10.1145/2830544.2830547

[17] Stefanie Beyer, Christian Macho, Martin Pinzger, and Massimiliano Di Penta. 2018. Automatically Classifying Posts into Question Categories on Stack Overflow. In *26th Conference on Program Comprehension, ICPC 2018, May 27-28, 2018, Gothenburg, Sweden.* 211–221. https://doi.org/10.1145/3196321.3196333

[18] Stefanie Beyer and Martin Pinzger. 2014. A Manual Categorization of Android App Development Issues on Stack Overflow. In *30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014, September 29 - October 3, 2014, Victoria, BC, Canada.* 531–535. https://doi.org/10.1109/ICSME.2014.88

[19] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to Ask for Technical Help? Evidence-based Guidelines for Writing Questions on Stack Overflow. *Inf. Softw. Technol.* 94 (2018), 186–207. https://doi.org/10.1016/j.infsof.2017.10.009

[20] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. 2021. Automated Query Reformulation for Efficient Search based on Query Logs From Stack Overflow. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, 22-30 May 2021, Madrid, Spain.* IEEE, 1273–1285. https://doi.org/10.1109/ICSE43902.2021.00116

[21] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2017. Using Observed Behavior to Reformulate Queries during Text Retrieval-based Bug Localization. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, September 17-22, 2017, Shanghai, China.* IEEE Computer Society, 376–387. https://doi.org/10.1109/ICSME.2017.100

[22] Oscar Chaparro, Juan Manuel Florez, and Andrian Marcus. 2019. Using Bug Descriptions to Reformulate Queries during Text-retrieval-based Bug Localization. *Empir. Softw. Eng.* 24, 5 (2019), 2947–3007. https://doi.org/10.1007/s10664-018-9672-z

[23] Oscar Chaparro, Juan Manuel Florez, Unnati Singh, and Andrian Marcus. 2019. Reformulating Queries for Duplicate Bug Report Detection. In *26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, February 24-27, 2019, Hangzhou, China.* IEEE, 218–229. https://doi.org/10.1109/SANER.2019.8667985

[24] Jinho D. Choi, Joel R. Tetreault, and Amanda Stent. 2015. It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers.* 387–396. https://doi.org/10.3115/v1/p15-1038

[25] Zachary Eberhart, Aakash Bansal, and Collin Mcmillan. 2020. A Wizard of Oz Study Simulating API Usage Dialogues with a Virtual Assistant. *IEEE Trans. Software Eng.* (2020). https://doi.org/10.1109/TSE.2020.3040935

[26] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic Query Reformulations for Text Retrieval in Software Engineering. In *35th International Conference on Software Engineering, ICSE 2013, May 18-26, 2013, San Francisco, CA, USA.* IEEE Computer Society, 842–851. https://doi.org/10.1109/ICSE.2013.6606630

[27] Abram Handler, Matthew Denny, Hanna M. Wallach, and Brendan O'Connor. 2016. Bag of What? Simple Noun Phrase Extraction for Text Analysis. In *1st Workshop on NLP and Computational Social Science, NLP+CSS@EMNLP 2016, November 5, 2016, Austin, TX, USA.* Association for Computational Linguistics, 114–124. https://doi.org/10.18653/v1/W16-5615

[28] Yasuhiro Hayase, Yu Kashima, Yuki Manabe, and Katsuro Inoue. 2011. Building Domain Specific Dictionaries of Verb-Object Relation from Source Code. In *15th European Conference on Software Maintenance and Reengineering, CSMR 2011, 1-4 March 2011, Oldenburg, Germany.* 93–100. https://doi.org/10.1109/CSMR.2011.15

[29] Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. 2009. Automatically Capturing Source Code Context of NL-queries for Software Maintenance and Reuse. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada.* IEEE, 232–242. https://doi.org/10.1109/ICSE.2009.5070524

[30] Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. 2011. Improving source code search with natural language phrasal representations of method signatures. In *26th IEEE/ACM International Conference on Automated Software Engineering ASE 2011, November 6-10, 2011, Lawrence, KS, USA*, Perry Alexander, Corina S. Pasareanu, and John G. Hosking (Eds.). IEEE Computer Society, 524–527. https://doi.org/10.1109/ASE.2011.6100115

[31] Yuki Kashiwabara, Yuya Onizuka, Takashi Ishio, Yasuhiro Hayase, Tetsuo Yamamoto, and Katsuro Inoue. 2014. Recommending Verbs for Rename Method using Association Rule Mining. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, February 3-6, 2014, Antwerp, Belgium.* 323–327. https://doi.org/10.1109/CSMR-WCRE.2014.6747186

[32] Niraj Kumar and Kannan Srinathan. 2008. Automatic keyphrase extraction from scientific documents using N-gram filtration technique. In *Proceedings of the 2008 ACM Symposium on Document Engineering, Sao Paulo, Brazil, September 16-19, 2008.* ACM, 199–208. https://doi.org/10.1145/1410140.1410180

[33] Hongwei Li, Sirui Li, Jiamou Sun, Zhenchang Xing, Xin Peng, Mingwei Liu, and Xuejiao Zhao. 2018. Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph. In *34th IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, September 23-29, 2018, Madrid, Spain.* 183–193. https://doi.org/10.1109/ICSME.2018.00028

[34] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. 2013. What Help do Developers Seek, When and How?. In *20th Working Conference on Reverse Engineering, WCRE 2013, October 14-17, 2013, Koblenz, Germany.* IEEE Computer Society, 142–151. https://doi.org/10.1109/WCRE.2013.6671289

[35] Zhixing Li, Tao Wang, Yang Zhang, Yun Zhan, and Gang Yin. 2016. Query Reformulation by Leveraging Crowd Wisdom for Scenario-based Software Search. In *8th Asia-Pacific Symposium on Internetware, Internetware 2016, September 18, 2016, Beijing, China.* ACM, 36–44. https://doi.org/10.1145/2993717.2993723

[36] Jiakun Liu, Sebastian Baltes, Christoph Treude, David Lo, Yun Zhang, and Xin Xia. 2021. Characterizing Search Activities on Stack Overflow. In *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, August 23-28, 2021, Athens, Greece.* ACM, 919–931. https://doi.org/10.1145/3468264.3468582

[37] Mingwei Liu, Xin Peng, Qingtao Jiang, Andrian Marcus, Junwen Yang, and Wenyun Zhao. 2018. Searching StackOverflow Questions with Multi-Faceted Categorization. In *10th Asia-Pacific Symposium on Internetware, Internetware 2018, September 16-16, 2018, Beijing, China.* ACM, 10:1–10:10. https://doi.org/10.1145/3275219.3275227

[38] Mingwei Liu, Xin Peng, Andrian Marcus, Shuangshuang Xing, Christoph Treude, and Chengyuan Zhao. 2021. API-Related Developer Information Needs in Stack Overflow. *IEEE Transactions on Software Engineering* (2021).

[39] Mingwei Liu, Xin Peng, Andrian Marcus, Zhenchang Xing, Wenkai Xie, Shuangshuang Xing, and Yang Liu. 2019. Generating Query-specific Class API Summaries. In *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia.* ACM, 120–130. https://doi.org/10.1145/3338906.3338971

[40] Yang Liu, Mingwei Liu, Xin Peng, Christoph Treude, Zhenchang Xing, and Xiaoxin Zhang. 2020. Generating Concept based API Element Comparison Using a Knowledge Graph. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, September 21-25, 2020, Melbourne, Australia.* IEEE, 834–845. https://doi.org/10.1145/3324884.3416628

[41] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query Expansion via WordNet for Effective Code Search. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, March 2-6, 2015, Montreal, QC, Canada.* IEEE Computer Society, 545–549. https://doi.org/10.1109/SANER.2015.7081874

[42] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query Expansion via WordNet for Effective Code Search. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, March 2-6,2015, Montreal, QC, Canada*. IEEE Computer Society, 545–549. https://doi.org/10.1109/SANER.2015.7081874

[43] Meili Lu, Xiaobing Sun, Shaowei Wang, David Lo, and Yucong Duan. 2015. Query expansion via WordNet for effective code search. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, March 2-6, 2015, Montreal, QC, Canada*. IEEE Computer Society, 545–549. https://doi.org/10.1109/SANER.2015.7081874

[44] Claudio Lucchese, Franco Maria Nardini, Raffaele Perego, Roberto Trani, and Rossano Venturini. 2018. Efficient and Effective Query Expansion for Web Search. In *27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*. ACM, 1551–1554. https://doi.org/10.1145/3269206.3269305

[45] Andrian Marcus, Andrey Sergeyev, Václav Rajlich, and Jonathan I. Maletic. 2004. An Information Retrieval Approach to Concept Location in Source Code. In *11th Working Conference on Reverse Engineering, WCRE 2004, November 8-12, 2004, Delft, The Netherlands*. IEEE Computer Society, 214–223. https://doi.org/10.1109/WCRE.2004.10

[46] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia Medica: Biochemia Medica* 22, 3 (2012), 276–282.

[47] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What Makes a Good Code Example?: A Study of Programming Q&A in StackOverflow. In *28th IEEE International Conference on Software Maintenance, ICSM 2012, September 23-28, 2012, Trento, Italy*. IEEE Computer Society, 25–34. https://doi.org/10.1109/ICSM.2012.6405249

[48] Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Trans. Serv. Comput.* 9, 5 (2016), 771–783. https://doi.org/10.1109/TSC.2016.2560165

[49] Francisca Pérez, Jaime Font, Lorena Arcega, and Carlos Cetina. 2019. Collaborative Feature Location in Models through Automatic Query Expansion. *Autom. Softw. Eng.* 26, 1 (2019), 161–202. https://doi.org/10.1007/s10515-019-00251-9

[50] Luca Ponzanelli, Andrea Mocci, Alberto Bacchelli, Michele Lanza, and David Fullerton. 2014. Improving Low Quality Stack Overflow Post Detection. In *30th IEEE International Conference on Software Maintenance and Evolution, ICSME 2014, September 29 - October 3, 2014, Victoria, BC, Canada*. IEEE Computer Society, 541–544. https://doi.org/10.1109/ICSME.2014.90

[51] Md. Masudur Rahman, Jed Barson, Sydney Paul, Joshua Kayani, Federico Andres Lois, Sebastian Fernandez Quezada, Christopher Parnin, Kathryn T. Stolee, and Baishakhi Ray. 2018. Evaluating How Developers Use General-purpose Web-search for Code Retrieval. In *15th International Conference on Mining Software Repositories, MSR 2018, May 28-29, 2018, Gothenburg, Sweden*. ACM, 465–475. https://doi.org/10.1145/3196398.3196425

[52] Mohammad Masudur Rahman and Chanchal K. Roy. 2016. QUICKAR: Automatic Query Reformulation for Concept Location using Crowdsourced Knowledge. In *31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, September 3-7, 2016, Singapore*. ACM, 220–225. https://doi.org/10.1145/2970276.2970362

[53] Mohammad Masudur Rahman and Chanchal K. Roy. 2017. Improved Query Reformulation for Concept Location using CodeRank and Document Structures. In *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*. IEEE Computer Society, 428–439. https://doi.org/10.1109/ASE.2017.8115655

[54] Mohammad Masudur Rahman and Chanchal K. Roy. 2018. Effective Reformulation of Query for Code Search Using Crowdsourced Knowledge and Extra-Large Data Analytics. In *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, September 23-29, 2018, Madrid, Spain*. IEEE Computer Society, 473–484. https://doi.org/10.1109/ICSME.2018.00057

[55] Sujith Ravi, Bo Pang, Vibhor Rastogi, and Ravi Kumar. 2014. Great Question! Question Quality in Community Q&A. In *8th International Conference on Weblogs and Social Media, ICWSM 2014,June 1-4, 2014, Ann Arbor, Michigan, USA*. The AAAI Press. http://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8080

[56] Manuel Roldan-Vega, Greg Mallet, Emily Hill, and Jerry Alan Fails. 2013. CONQUER: A Tool for NL-Based Query Refinement and Contextualizing Code Search Results. In *2013 IEEE International Conference on Software Maintenance, ICSM 2013, September 22-28, 2013, Eindhoven, The Netherlands*. IEEE Computer Society, 512–515. https://doi.org/10.1109/ICSM.2013.84

[57] Christoffer Rosen and Emad Shihab. 2016. What are Mobile Developers Asking about? A Large Scale Study using Stack Overflow. *Empir. Softw. Eng.* 21, 3 (2016), 1192–1223. https://doi.org/10.1007/s10664-015-9379-3

[58] David C. Shepherd, Zachary P. Fry, Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. 2007. Using Natural Language Program Analysis to Locate and Understand Action-oriented Concerns. In *6th International Conference on Aspect-Oriented Software Development, AOSD 2007, March 12-16, 2007, Vancouver, British Columbia, Canada (ACM International Conference Proceeding Series, Vol. 208)*. ACM, 212–224. https://doi.org/10.1145/1218563.1218587

[59] Ravindra Singh and Naurang Singh Mangat. 2013. *Elements of Survey Sampling*. Vol. 15. Springer Science & Business Media.

[60] Bunyamin Sisman and Avinash C. Kak. 2013. Assisting Code Search with Automatic Query Reformulation for Bug Localization. In *10th Working Conference on Mining Software Repositories, MSR 2013, May 18-19, 2013, San Francisco, CA, USA*. IEEE Computer Society, 309–318. https://doi.org/10.1109/MSR.2013.6624044

[61] Sampath Srinivas. 1993. A Generalization of the Noisy-Or Model. In *9th Annual Conference on Uncertainty in Artificial Intelligence, UAI 1993, July 9-11, 1993, The Catholic University of America, Providence, Washington, DC, USA*. Elsevier, 208–215.

[62] Jiamou Sun, Zhenchang Xing, Rui Chu, Heilai Bai, Jinshui Wang, and Xin Peng. 2019. Know-How in Programming Tasks: From Textual Tutorials to Task-Oriented Knowledge Graph. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, September 29 - October 4, 2019, Cleveland, OH, USA*. IEEE, 257–268. https://doi.org/10.1109/ICSME.2019.00039

[63] Yuan Tian, Ferdian Thung, Abhishek Sharma, and David Lo. 2017. APIBot: question answering bot for API documentation. In *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*. IEEE Computer Society, 153–158. https://doi.org/10.1109/ASE.2017.8115628

[64] Christoph Treude, Ohad Barzilay, and Margaret-Anne D. Storey. 2011. How do Programmers Ask and Answer Questions on the Web?. In *33rd International Conference on Software Engineering, ICSE 2011, May 21-28, 2011, Waikiki, Honolulu, HI, USA*. ACM, 804–807. https://doi.org/10.1145/1985793.1985907

[65] Christoph Treude, Martin P. Robillard, and Barthélémy Dagenais. 2015. Extracting Development Tasks to Navigate Software Documentation. *IEEE Trans. Software Eng.* 41, 6 (2015), 565–581. https://doi.org/10.1109/TSE.2014.2387172

[66] Pradeep K. Venkatesh, Shaohua Wang, Ying Zou, and Joanna W. Ng. 2017. A Personalized Assistant Framework for Service Recommendation. In *2017 IEEE International Conference on Services Computing, SCC 2017, June 25-30, 2017, Honolulu, HI, USA*. IEEE Computer Society, 92–99. https://doi.org/10.1109/SCC.2017.20

[67] Chong Wang, Xin Peng, Mingwei Liu, Zhenchang Xing, Xuefang Bai, Bing Xie, and Tuo Wang. 2019. A Learning-Based Approach for Automatic Construction of Domain Glossary from Source Code and Documentation. In *27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*. ACM, 97–108. https://doi.org/10.1145/3338906.3338963

[68] Lu Wang, Xiaobing Sun, Jingwei Wang, Yucong Duan, and Bin Li. 2017. Construct Bug Knowledge Graph for Bug Resolution: Poster. In *39th International Conference on Software Engineering, ICSE 2017 - Companion Volume, May 20-28, 2017 , Buenos Aires, Argentina*. IEEE Computer Society, 189–191. https://doi.org/10.1109/ICSE-C.2017.102

[69] Bernard L Welch. 1947. The Generalization of Student's Problem when Several Different Population Variances are Involved. *Biometrika* 34, 1/2 (1947), 28–35.

[70] Wenkai Xie, Xin Peng, Mingwei Liu, Christoph Treude, Zhenchang Xing, Xiaoxin Zhang, and Wenyun Zhao. 2020. API method recommendation via explicit matching of functionality verb phrases. In *28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2020, November 8-13, 2020, Virtual Event, USA*. ACM, 1015–1026.

[71] Shuangshuang Xing, Mingwei Liu, and Xin Peng. 2021. Automatic Code Semantic Tag Generation based on Software Knowledge Graph. *Journal of Software* (2021).

[72] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated Generation of Answer Summary to Developersź Technical Questions. In *32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, October 30 - November 03, 2017, Urbana, IL, USA*. 706–716. https://doi.org/10.1109/ASE.2017.8115681

[73] Neng Zhang, Qiao Huang, Xin Xia, Ying Zou, David Lo, and Zhenchang Xing. 2022. Chatbot4QR: Interactive Query Refinement for Technical Question Retrieval. *IEEE Trans. Software Eng.* 48, 4 (2022), 1185–1211. https://doi.org/10.1109/TSE.2020.3016006

[74] Neng Zhang, Jian Wang, Yutao Ma, Keqing He, Zheng Li, and Xiaoqing (Frank) Liu. 2018. Web Service Discovery based on Goal-oriented Query Expansion. *J. Syst. Softw.* 142 (2018), 73–91. https://doi.org/10.1016/j.jss.2018.04.046